Axiom Forms and Dashboards Guide

Axiom Software

Version 2018.2

KaufmanHall

axiom software

Kaufman Hall
Axiom Software
10260 SW Greenburg Road, Suite 710
Portland, Oregon 97223
877.691.9969 (Toll-Free)
503.977.0234 (Local)
www.kaufmanhall.com

Table of Contents

Introduction

Using the Axiom forms feature in Axiom Software, you can create rich web-enabled files for data collection, home pages, and reporting. Axiom forms can be viewed within all Axiom Clients, but when viewed using the Axiom Web Client it includes support for viewing forms via devices such as the iPad or an Android tablet.

Axiom Software also provides a set of robust data visualization tools for use in Axiom forms, including bar / column / line charts, pie charts, scatter and bubble charts, various gauges, and drawing tools. These data visualization tools are also known as "dashboard" components.

Intended audience

This guide is intended for administrators and other power users who create Axiom forms for end users.

What is covered in this guide?

This guide covers the following aspects of Axiom forms:

- Enabling files for Axiom forms
- Designing the form canvas and using various components
- Setting up interactive components
- Saving data from Axiom forms
- Using dashboard components in Axiom forms
- Publishing forms to end users
- Using Axiom forms as custom dialogs and task panes
- Configuring options for form display in the Web Client

What is not covered in this guide?

The following related topics are not covered in this guide:

- Creating Axiom files, including using Axiom queries and Axiom functions. For more information, see the *Axiom File Setup Guide*.

All documentation for Axiom Software can also be accessed using the Axiom Software Help Files.

Axiom Software Client versions

This guide discusses functionality that is available in the Axiom Desktop Client (Excel Client and Windows Client). Screenshots of features may show either the Excel Client or the Windows Client. The Axiom Software functionality is virtually identical in both environments.

Axiom forms are built within the Desktop Client, and then can be viewed in any client, including the Axiom Web Client. The Axiom Web Client is only for viewing completed forms; no form setup activities are available in this client.

Axiom Forms Overview

Using Axiom forms, you can create rich, web-enabled files for users to interact with on any Axiom client. These forms can be viewed within the Excel Client or the Windows Client, as well as the Web Client—including support for viewing via devices such as the iPad or an Android tablet.

The basic Axiom forms feature allows you to:

- Present data in a formatted grid view
- Use interactive controls to change data views and/or collect user inputs
- Save user inputs to the database
- Use other Axiom features such as executing a Scheduler job from the form or creating a new on-demand plan file

Axiom Software also provides a set of robust data visualization tools for use in Axiom forms, including bar / column / line charts, pie charts, scatter and bubble charts, various gauges, and drawing tools. These data visualization tools are also known as "dashboard" components.

NOTE: The ability to create and edit Axiom forms is controlled by your Axiom Software license. For more information, see Licensing requirements for Axiom forms.

Axiom forms are created using Axiom files, such as report files or templates / plan files. The form is designed within the Axiom file, using a special Form Control Sheet in conjunction with the Form Assistant task pane and the Form Designer.

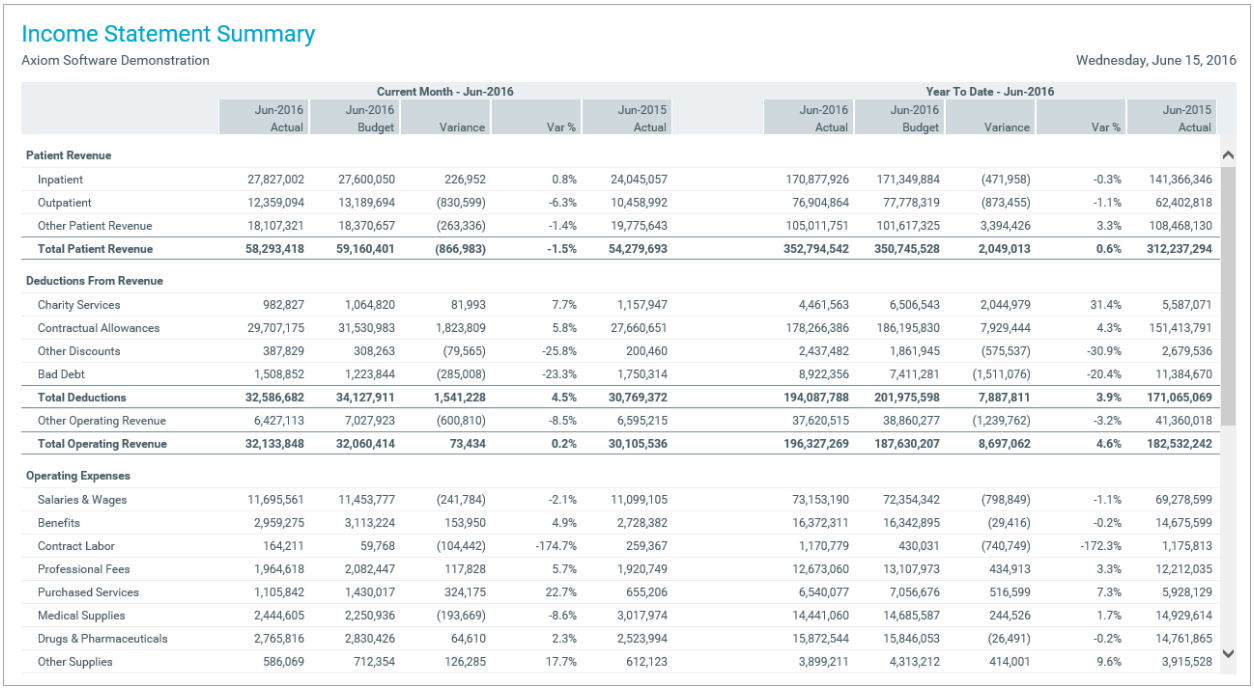
Essentially, when you design an Axiom form you create a web "view" of that file. The design process consists of placing various form *components* on the form *canvas*, and then configuring various properties for each component. To display data within the form, you bring the data in the Axiom file using normal query methods—such as Axiom queries—and then link the data in the file to the components on the form canvas.

Once the file is configured as desired, users can view the file as a web form. The web form is generated "on the fly" for each user, based on the form settings in the source file.

Users can view and interact with the components in the form, including the ability to make interactive selections that impact the data displayed in the form. For example, users could select something from a drop-down list to indicate what data they want to see in a particular formatted grid or chart. Data queries are made live to the database so that the form always displays the latest data, based on the user's

security filters. You can also configure a form to accept user inputs that are then saved back to the Axiom database via the source file.

The following screenshots show examples of various Axiom forms.



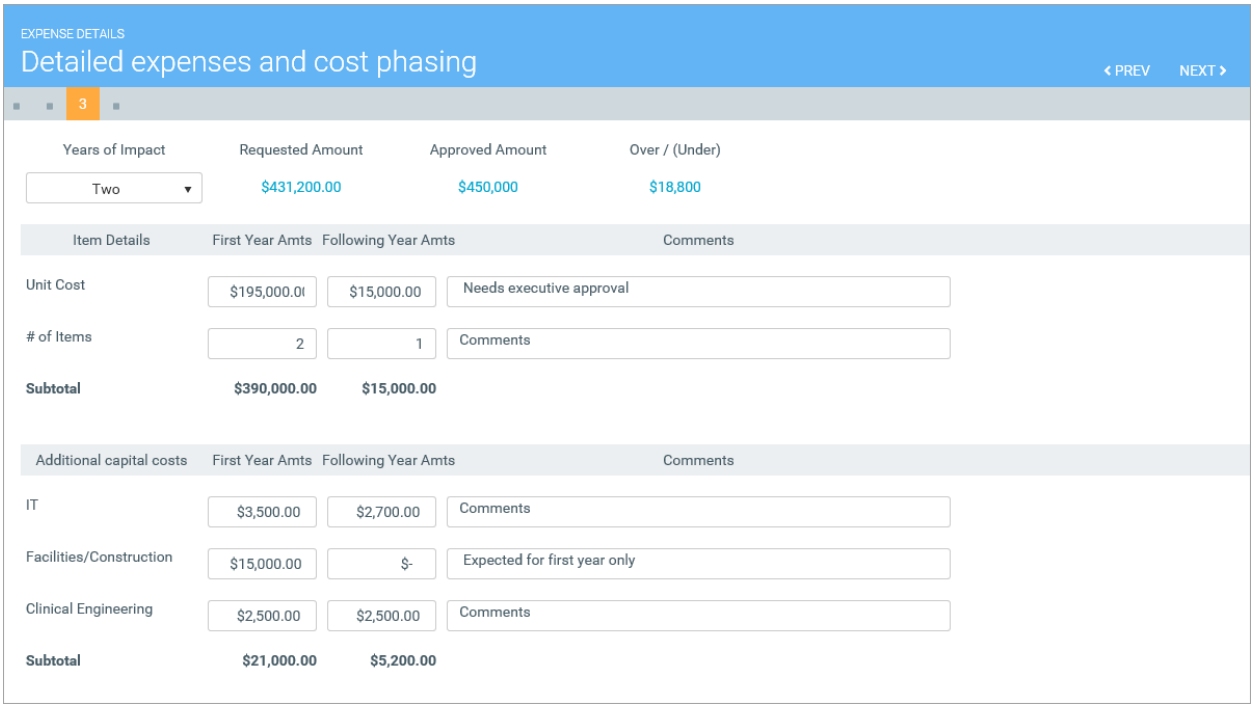
Income Statement Summary

Axiom Software Demonstration

Wednesday, June 15, 2016

	Current Month - Jun-2016					Year To Date - Jun-2016				
	Jun-2016 Actual	Jun-2016 Budget	Variance	Var %	Jun-2015 Actual	Jun-2016 Actual	Jun-2016 Budget	Variance	Var %	Jun-2015 Actual
Patient Revenue										
Inpatient	27,827,002	27,600,050	226,952	0.8%	24,045,057	170,877,926	171,349,884	(471,958)	-0.3%	141,366,346
Outpatient	12,359,094	13,189,694	(830,599)	-6.3%	10,458,992	76,904,864	77,778,319	(873,455)	-1.1%	62,402,818
Other Patient Revenue	18,107,321	18,370,657	(263,336)	-1.4%	19,775,643	105,011,751	101,617,325	3,394,426	3.3%	108,468,130
Total Patient Revenue	**58,293,418**	**59,160,401**	**(866,983)**	**-1.5%**	**54,279,693**	**352,794,542**	**350,745,528**	**2,049,013**	**0.6%**	**312,237,294**
Deductions From Revenue										
Charity Services	982,827	1,064,820	81,993	7.7%	1,157,947	4,461,563	6,506,543	2,044,979	31.4%	5,587,071
Contractual Allowances	29,707,175	31,530,983	1,823,809	5.8%	27,660,651	178,266,386	186,195,830	7,929,444	4.3%	151,413,791
Other Discounts	387,829	308,263	(79,565)	-25.8%	200,460	2,437,482	1,861,945	(575,537)	-30.9%	2,679,536
Bad Debt	1,508,852	1,223,844	(285,008)	-23.3%	1,750,314	8,922,356	7,411,281	(1,511,076)	-20.4%	11,384,670
Total Deductions	**32,586,682**	**34,127,911**	**1,541,228**	**4.5%**	**30,769,372**	**194,087,788**	**201,975,598**	**7,887,811**	**3.9%**	**171,065,069**
Other Operating Revenue	6,427,113	7,027,923	(600,810)	-8.5%	6,595,215	37,620,515	38,860,277	(1,239,762)	-3.2%	41,360,018
Total Operating Revenue	**32,133,848**	**32,060,414**	**73,434**	**0.2%**	**30,105,536**	**196,327,269**	**187,630,207**	**8,697,062**	**4.6%**	**182,532,242**
Operating Expenses										
Salaries & Wages	11,695,561	11,453,777	(241,784)	-2.1%	11,099,105	73,153,190	72,354,342	(798,849)	-1.1%	69,278,599
Benefits	2,959,275	3,113,224	153,950	4.9%	2,728,382	16,372,311	16,342,895	(29,416)	-0.2%	14,675,599
Contract Labor	164,211	59,768	(104,442)	-174.7%	259,367	1,170,779	430,031	(740,749)	-172.3%	1,175,813
Professional Fees	1,964,618	2,082,447	117,828	5.7%	1,920,749	12,673,060	13,107,973	434,913	3.3%	12,212,035
Purchased Services	1,105,842	1,430,017	324,175	22.7%	655,206	6,540,077	7,056,676	516,599	7.3%	5,928,129
Medical Supplies	2,444,605	2,250,936	(193,669)	-8.6%	3,017,974	14,441,060	14,685,587	244,526	1.7%	14,929,614
Drugs & Pharmaceuticals	2,765,816	2,830,426	64,610	2.3%	2,523,994	15,872,544	15,846,053	(26,491)	-0.2%	14,761,865
Other Supplies	586,069	712,354	126,285	17.7%	612,123	3,899,211	4,313,212	414,001	9.6%	3,915,528

Example report



EXPENSE DETAILS

Detailed expenses and cost phasing

‹ PREV NEXT ›

3

Years of Impact	Requested Amount	Approved Amount	Over / (Under)
Two	$431,200.00	$450,000	$18,800

Item Details	First Year Amts	Following Year Amts	Comments
Unit Cost	$195,000.0(	$15,000.00	Needs executive approval
# of Items	2	1	Comments
Subtotal	**$390,000.00**	**$15,000.00**	

Additional capital costs	First Year Amts	Following Year Amts	Comments
IT	$3,500.00	$2,700.00	Comments
Facilities/Construction	$15,000.00	$-	Expected for first year only
Clinical Engineering	$2,500.00	$2,500.00	Comments
Subtotal	**$21,000.00**	**$5,200.00**	

Example input form / plan file

Example dashboard

How Axiom forms are rendered to users

When a user opens an Axiom file as an Axiom form, a copy of the source file is opened on the Axiom Application Server using the Web spreadsheet engine. The Axiom form settings in the source file are read by the application server and rendered as a web page using the HTML5 markup language. The user's browser must support HTML5 in order to properly view the Axiom form and use form functionality. For more information on the browser requirements for viewing Axiom forms, see the *Technical Requirements* document for Axiom Software.

The end user viewing the Axiom form does not see the source file; they only see the web form. The source file is used to query the data to be provided to the Axiom form, and to support other form functionality such as component interactivity and saving to the database. The source file is opened read-only and is not locked to the user—other users can also open the file as an Axiom form, and the source file is still eligible to opened read/write in the Desktop Client by a user with the appropriate rights.

When the user first opens the Axiom form, the source file is refreshed and the initial state of the form is determined. The Axiom form is subsequently updated in response to changes to interactive components (if set to Auto-Submit) or if the user clicks a Button component. This behavior is described in more detail in Update and save behavior for Axiom forms. The form contents can also be changed by using the filter panel—see Defining refresh variables for the Web Client Filters panel.

Certain events will cause the Axiom form to "reset," thereby causing the user to lose the current state of the form (including any unsaved data):

- The user closes the browser window or the tab for the form, or navigates to another page (within the same tab).
- The user refreshes the Axiom form using the browser functionality (for example, pressing F5) instead of by using a Button component.
- The user reloads the Axiom form from the address bar (for example, the user places their cursor in the address bar and presses Enter to reload), or the user clicks a hyperlink to the same Axiom form from within another file or form.
- The network connection to the Axiom Application Server is interrupted for over five minutes. In this case the forms session is lost along with the current state of the form.

In the first three cases, if the Axiom form is configured to save data and unsaved data is detected, the user will be warned about the unsaved data and prompted to confirm that they want to continue. This warning only applies to data that has been submitted from the Axiom form to the source file but not yet saved to the database—unsubmitted changes in the form web page will not trigger the warning. Note that this warning is not available when using Axiom forms on an iPad.

Axiom forms creation process

The following is an overview of the Axiom forms creation process.

Visualizing your form

Before beginning any Axiom forms creation in Axiom Software, you should spend a few moments visualizing what you want the end result to look like. Ask yourself questions such as:

- What data do you want users to see, and how do you want them to see it?
- What is the primary goal of the form? For example, is it primarily a report that displays data, or is it primarily an input form to gather data?.
- Do you want the data in the form to be interactive? If so, what do you want users to be able to change, and how do you want users to be able to make these selections? Can the selections be handled by using refresh variables, or do you need to set up interactive components on the form itself?

- If the form will be used to save data to the database, does the form also need to be able to retrieve saved data? You will need to think through the file setup to accommodate saving data and displaying current values.
- How will users view the form? Will they be viewing on a full-screen browser on their client workstations, or using a tablet computer?

It may be helpful to begin by sketching the desired end result on a piece of paper.

Process steps

The basic process steps for creating an Axiom form are as follows:

1. Create a forms-enabled file. This file will be the source file for the Axiom form. See Enabling a file for Axiom forms.
2. Set up the source file as needed to query the desired data for the Axiom form. You can use Axiom queries and Axiom functions to return data, and you can "hard-code" data within the spreadsheet. See Setting up the source file for the Axiom form.
3. Design the canvas for the Axiom form with the desired components. See the following:
   - Designing the Form Canvas
   - Axiom Form Components
4. Configure the components for the Axiom form. See the following:
   - Linking components to data
   - Using interactive components in an Axiom form
5. Configure any other special features for the Axiom form, such as saving to the database or performing various actions. See General Design Concepts for Axiom Forms and Using Axiom Forms for Planning.
6. Preview the Axiom form to test how the form will display to end users. See Previewing an Axiom form.
7. Determine how users will access the finished Axiom form. See Publishing Axiom Forms.

Enabling a file for Axiom forms

Axiom forms are created by using Axiom files. To create an Axiom form, you enable the Axiom file for form development by adding a Form Control Sheet.

The first step is to determine which type of file to use as your source file for the Axiom form. Report files and file group files can be enabled for Axiom forms. For file groups, the most common use case is to enable a template for Axiom forms in order to generate form-enabled plan files. However, you can also enable Axiom forms for individual plan files, driver files, and utilities if needed.

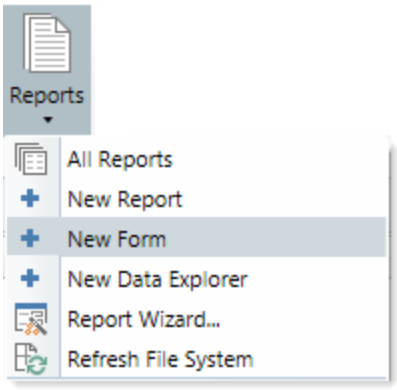
Once a file has been enabled for Axiom forms, the **Form Assistant** task pane becomes available to help you configure the form. The file is now flagged as an Axiom forms file in addition to its native file type (report, plan file, etc.). This means that certain features are now available for that file—such as the ability to open it as an Axiom form in a browser—and the file will display in the list of Axiom forms on the Axiom Forms Server web page (if a user has rights to the file).

NOTE: In general, if an Axiom file is form-enabled, then that file should be dedicated to Axiom forms use only. This means that end users should only interact with the file as an Axiom form—you should not attempt to configure a file so that some users access it as a form and some users access it as a spreadsheet. There are no technical limitations in this regard; it is simply easier to maintain the file and grant user access if the file is limited to Axiom forms use. For optimal performance, the source file for an Axiom form should only contain the features and data relevant to the form—unnecessary content in the file can slow web performance and complicate file maintenance.

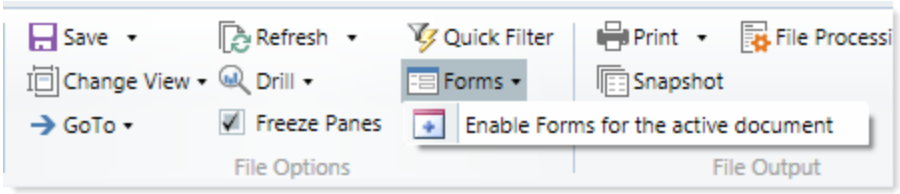
Creating Axiom forms using report files

You can create a form-enabled report in any of the following ways:

- Create a new blank report with a Form Control Sheet. From the **Reports** menu, click **New Form**.



- Open an existing report and then manually add a Form Control Sheet. In the **File Options** group, click **Forms > Enable Forms for the active document**.



- Create a new report using the Report Wizard. On the **Report Options** screen of the wizard, select **Enable form authoring**. When the wizard creates your report, it will also add a Form Control Sheet.

You can save the form-enabled report file to any area of the Reports Library. Keep in mind that users must have at least read-only access to the file in order to view the finished form. You may want to create a folder specifically for form-enabled files and then grant access to that folder as appropriate.

NOTE: In order to create a new form-enabled report, users must have read/write permission AND Sheet Assistant permission to at least one folder. In order to enable forms for an existing report, users must have Sheet Assistant permission to that report.

Creating Axiom forms for templates and other file group files

For all file group files, you can enable the file for Axiom forms using the **Forms** menu item in the **File Options** group. Click **Forms > Enable Forms for the active document**.

In most cases if you want to create form-enabled plan files, you will enable forms at the template level, however you can enable forms at the individual plan file level if needed.

NOTE: In order to enable forms for an existing file, users must have Sheet Assistant permission to that file.

Licensing requirements for Axiom forms

Your Axiom Software license determines the level of features available to you to create Axiom forms. There are three levels of licensing:

- **Standard**: The Form Designer is not available to create new forms.
- **Limited Forms**: The Form Designer is available to create forms using standard components only. Dashboard components are not available in the Form Designer.
- **Full Forms**: The Form Designer is available to create forms using all components, including standard components and dashboard components. Dashboard components can be used to create interactive and visual dashboards using the Axiom forms web-based technology.

All licenses include the ability to view already-created Axiom forms, such as forms and dashboards that are provided with a packaged product. The license simply controls access to form creation, by restricting or enabling access to the Form Designer, and restricting which components are available in the Form Designer.

Dashboard components include various charts and graphs, gauges, and drawing tools. All components in the **Charts** section and the **Shapes** section of the Form Designer require a Full Forms license.

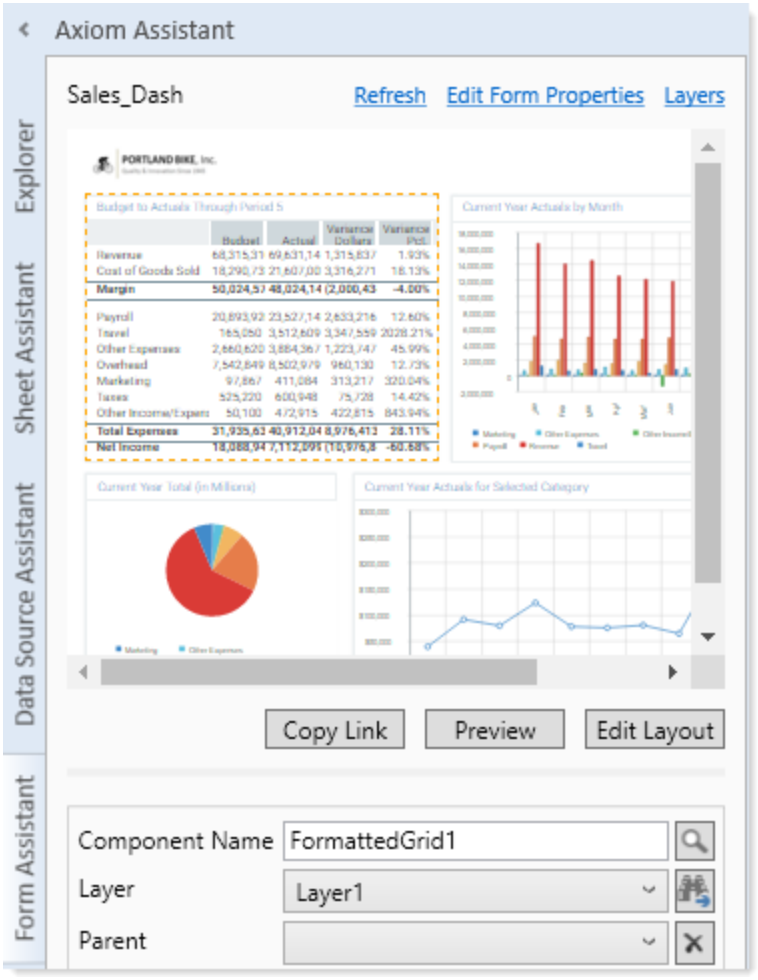
Dashboards are entirely created using the Axiom forms technology. To create a new dashboard, you create a new form and use the special dashboard components within that form. All information on creating, configuring, and publishing Axiom forms applies to dashboards.

If you are not sure what level of licensing you have for Axiom forms, please contact Kaufman Hall Software Support for more details and assistance.

Designing the Form Canvas

The Axiom form *canvas* is the area where you design what users will see on the form. You can view and edit the canvas by using the **Form Assistant** task pane and the **Form Designer** dialog.

The Form Assistant displays a thumbnail view of the canvas. From this area, you can open the canvas for editing, or you can preview the Axiom form. You can also edit the properties of existing components on the canvas.



Example Form Assistant task pane

When you click **Edit Layout**, the Form Designer dialog opens so that you can edit the form canvas. Using the Form Designer, you can add and remove components on the canvas, size and position components on the canvas, and configure component properties.

The Form Assistant is only available to administrators or to users with read/write access to the file AND Sheet Assistant permission. Additionally, your Axiom Software license determines whether the Form Assistant is available to your installation.

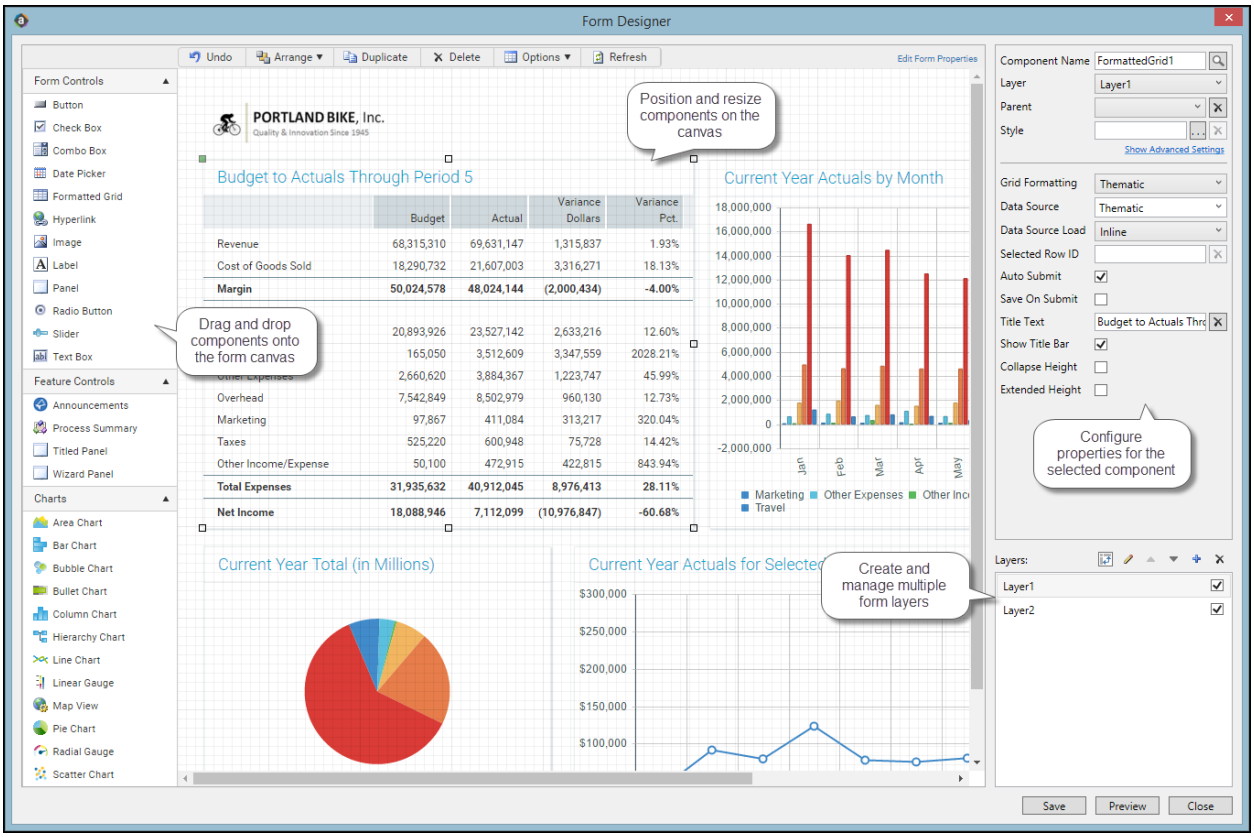
Using the Form Designer to design the form canvas

Axiom forms are created by dragging and dropping various components onto the form canvas using the Form Designer, and then configuring the data and display properties of those components.

To open the canvas for editing, do one of the following. The file must already be form-enabled in order to use these options.

- In the **Form Assistant** task pane, click the **Edit Layout** button. If the canvas is currently empty, you can also click the **Open layout editor** link that displays in the thumbnail area.
- On the **Axiom** ribbon tab, in the **File Options** group, click **Forms > Launch Form Designer**.

Either action opens the **Form Designer** dialog. The left-hand side of the dialog displays the available Axiom form components. The middle section is the canvas. When you select an item on the canvas, properties for that item display on the right-hand side of the dialog.



Example Form Designer dialog

The toolbar contains several commands to help arrange and design the components on the canvas, as described in the following sections. An **Undo** command is available to undo any change made to the canvas in the current session.

The form contents display using actual size and location within the Form Designer dialog. In order to view all of the form contents, you may need to scroll within the dialog or size the dialog larger.

The Form Designer displays the form contents as they will display in the form web page, with a few minor differences and limitations. The intent of the Form Designer is to provide you with a good idea of how the form will render, not to display a live working form. You should always view the form in its intended environment to truly test the design. Note the following display differences in the Form Designer:

- Interactive components are not interactive in this environment. For example, if you place a CheckBox component on the canvas, you will see the check box as it will display in the rendered form, but you cannot actually check or clear the check box.
- For components that use data sources, actual data is read into the component, but only enough to provide a reasonable representation of the component. Data does not update in real time, and some aspects of the component may not render as they will in a live form.

- If a component is configured as not visible (because a formula is being used to dynamically show or hide it), then that component still displays on the canvas in the Form Designer. This behavior is so that you can continue to work with the component on the canvas, without having to change or remove any formulas that determine the component's visibility in the rendered form. If you want to be able to hide and show components within the Form Designer, you can use layers. See Using multiple layers on the canvas.
- When you select a component on the canvas, that component will temporarily display "above" any other components. This is so that you can configure the component on the canvas without needing to adjust the component's rendering order (by moving to front or back). Once the component is no longer selected, it will revert to its actual rendering order. This behavior is only noticeable when you have components stacked on top of other components.

Adding, copying, and deleting components

You can add new components to the canvas, copy existing components, and delete components. For more information on the available component types and their properties, see Axiom Form Components.

- To add a component to the canvas, select the component type from the left-hand side of the dialog, and then drag and drop the component to the desired place on the canvas. You can add as many components as needed, including multiples of the same component type.

  When you add a component to the canvas, a section is automatically added to the Form Control Sheet to define the properties for that component. The properties also display in the right-hand side of the Form Designer dialog when the component is selected on the canvas.

  In most cases, you can define the component properties in the Form Designer or in the Form Assistant instead of editing the Form Control Sheet directly (unless you want to use a formula for that property, in which case you must edit the control sheet). If you want to manually edit the control sheet properties for a component, you can jump to the appropriate section in the control sheet by double-clicking on any property name—your cursor will be placed in that property for the current component.
- To copy an existing component, select the component that you want to copy, and then click **Duplicate**.
- To delete a component from the canvas, select the component and then click **Delete**. This will remove the component from the canvas and from the Form Control Sheet.

Moving and resizing components

Once a component has been added to the canvas, you can move it to different locations, and you can resize it.

- To move a component, select the component on the canvas and then drag and drop the component to the desired location. You can also use the arrow keys to move the component in the arrow direction (up, down, left, right).

- To resize a component, select the component on the canvas and then hover your cursor over one of the selection handles on the corners and sides of the component, so that the cursor becomes a two-sided arrow. You can then resize the component to a larger or smaller size by dragging the selection handle.

  To maintain the current proportions when you resize, hold down the SHIFT key and drag any of the non-corner selection handles. For example, if you want to resize an image but maintain the current aspect ratio.

- To make two components the same size, select the component that is the size you want to copy, and then select one or more additional components (using SHIFT or CTRL). Click **Arrange > Make Same Size**. All selected components will be resized to match the size of the first component.
- To move a component forward or backward on the canvas (within the current layer), select the component. Click **Arrange** and then select either **Send to Back in Layer** or **Bring to Front in Layer**. For example, if you select **Arrange > Send to Back in Layer**, then the selected component will be moved to the back of the canvas, underneath all other components in the layer.
- To move a component to another layer on the canvas, use **Arrange > Move to Layer**. For more information, see Using multiple layers on the canvas.

NOTE: The **Snap To Grid** option and the built-in alignment lines may impact the placement and size of a component when adjusting it on the canvas. See the following sections on these features for more information.

This section describes the simple behavior of setting component size and position by modifying the component on the canvas. When using this behavior, all components are positioned relative to the upper left corner of the canvas (or of the parent panel, if the component is a child of a panel), and all sizes and positions are defined in pixels. Alternatively, you can use the advanced component settings to define dynamic size and position options, such as percentage-based sizes and positions. For more information, see Controlling component position and size.

Using grid lines

By default, the canvas area displays with grid lines to help you size and position items on the canvas. Each minor line in the grid represents 10px and each major line in the grid represents 100px. The grid lines only display in the Form Designer; they will not display in the rendered form.

When moving and resizing components on the canvas, you can choose whether components snap to the nearest grid line. The "snap to grid" behavior is available regardless of whether the grid is currently shown on the canvas.

To configure grid options for the Form Designer:

- To toggle the grid lines on or off, click **Options > Show Grid**.
- To toggle the snap to grid behavior, click **Options > Snap To Grid**.

NOTE: When grid lines are shown, they display on top of components on the canvas. This behavior allows gridlines to be seen even when using form designs that include full-page panels, such as the Wizard Panel or the Titled Panel.

Aligning components

You can align components on the canvas with other components. To do this:

1. Select the component that you want to use as the "anchor" for the alignment.
2. Select one or more additional components (using SHIFT or CTRL).
3. Click **Arrange > Align > *AlignmentPoint***.

The components will be aligned with the "anchor" component, according to the selected alignment point. For example, if you click **Arrange > Align > Left**, then all selected components will be aligned with the left side of the "anchor" component.

When you move a component by dragging it on the canvas, Axiom will display red alignment lines when the component is close to becoming aligned with another component. If you release the component ("drop" it) when a red alignment line is visible, the component will be automatically aligned along that axis.

NOTES:

- If Snap to Grid is enabled and you move a component near another component that is not on a grid line, then the alignment guides will not display. If you want the alignment lines to show in this case, you can disable Snap to Grid. The Align command will work regardless of whether Snap to Grid is enabled.
- If components use differing units for position—for example, one uses pixels and one uses percentage, then alignment lines will not display for these components. If you use **Arrange > Align** to align these components, the components will be changed to use the units of the anchor component. For example, if the anchor component uses percentage and the other selected components use pixels, the other components will be changed to percentage.

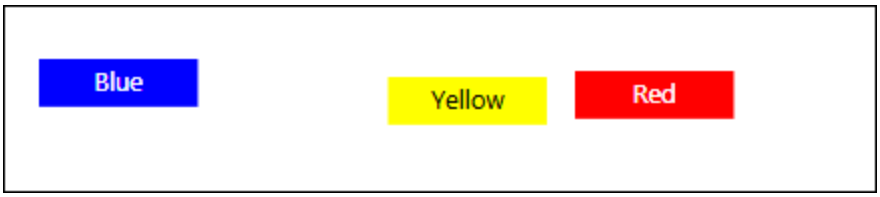
Distributing components

You can distribute three or more components evenly on the canvas, either horizontally or vertically. The components are distributed within the area bounded by the two outermost components (not across the entire width or height of the canvas). To distribute components:

1. Select three or more components on the canvas (using SHIFT or CTRL).
2. Click **Arrange > Align**, and then select either **Distribute Horizontally** or **Distribute Vertically**.

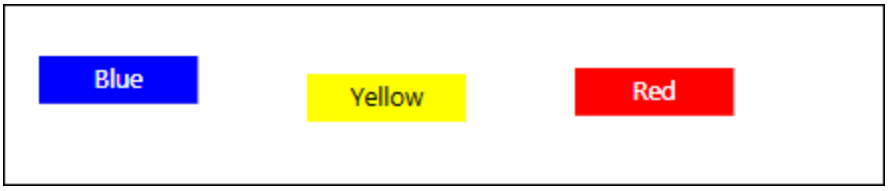
The components are distributed so that they are equidistant from each other.

For example, imagine that you have three labels placed on the canvas, and you want these labels to be equal distance from each other:



Before distribution

After selecting all three components and then clicking **Arrange > Align > Distribute Horizontally**, the labels would be evenly distributed as follows:



After distribution

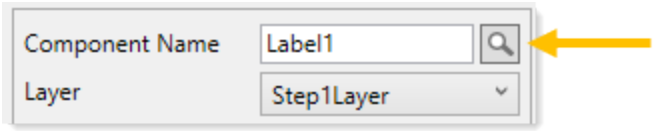
Notice that the components are distributed in the area defined from the edge of the blue component to the edge of the red component, not within the overall canvas width. Assuming these components should also be aligned with each other, you could then select the blue component (or whichever component was at the desired location) and then click **Arrange > Align > Top**.

Finding components

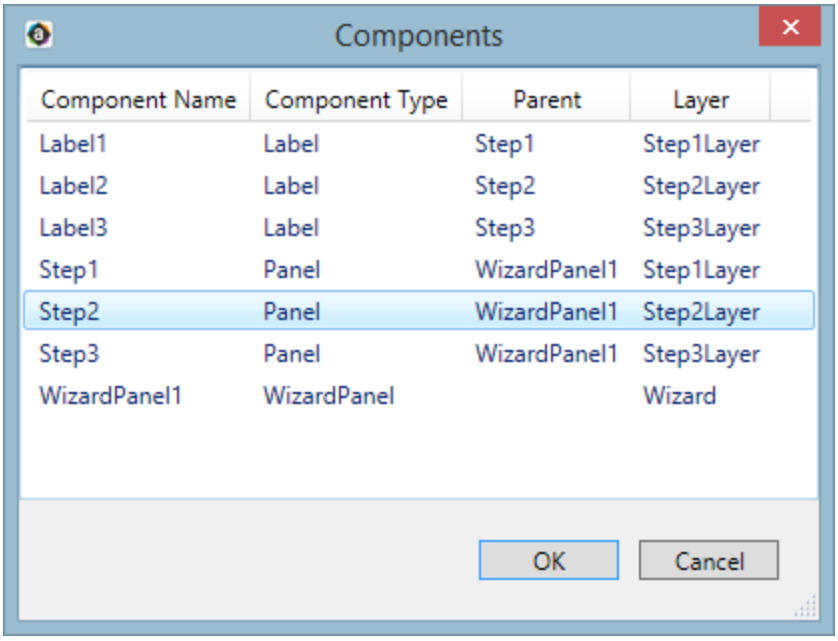
You can search for any component on the form canvas, so that its properties display in the Form Designer or Form Assistant. You might want to do this if the component is on a layer that is not currently shown in the Form Designer, or if there are just a lot of components on the form and it is difficult to find and select the component. Also, you might know the name or type of component that you want to edit, but you might not be sure where it is placed on the canvas.

To search for a component:

1. Click the Search button to the right of the **Component Name** box. You must already have a component selected in order to do this.



The **Components** dialog opens, displaying a list of all components in the form. Components are listed by name, type, parent, and layer. To sort based on any of these columns, click the column header.



2. Once you have located the component that you want to view, select that row and click **OK**.

The Form Designer and Form Assistant update to show the properties of the selected component. If the component is visible on the form canvas, it also becomes selected on the canvas.

Controlling component position and size

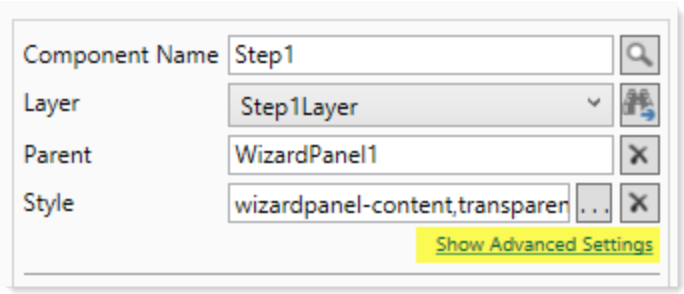
Axiom Software provides a variety of ways to control the position and size of components on an Axiom form, including options that allow components to dynamically adjust based on the current page size.

The default and most basic way to control position and size is to use the Form Designer to move and resize components on the canvas. When you drag and drop a component to a new location, Axiom Software automatically updates the position coordinates of the component for you. And when you drag a selection handle of a component to a larger or smaller size, Axiom Software automatically updates the width and height of the component to the new size.

However, often you need a more dynamic way to define the size or position of a component. For example, you may want a component to always be placed in the lower right corner or always fill 50% of the page width, regardless of the current page size. To do this, you can use the advanced component settings to adjust position and size.

To access the position and size settings for a component:

- In the Form Designer or Form Assistant, click **Show Advanced Settings** (located under the **Style** box). This exposes the position and size settings for the currently selected component.



You can also access the position and size settings for a component on the Form Control Sheet. Generally speaking, it is not recommended to edit the settings this way, because the canvas view will not automatically update for your changes.

The following settings are available to control component position and size. See the detailed discussions on each setting for more information.

Item	Description
Reference Location	The reference location determines how the x-position and y-position of a component are evaluated. By default the reference location is UpperLeft. **NOTE:** This setting is not exposed in the advanced component settings. It can be changed on the canvas by double-clicking the corner selection handles of a component, or you can edit the setting on the Form Control Sheet directly.
X Position Y Position	The x-position determines the component's position along the horizontal axis, and the y-position determines the component's position along the vertical axis. Both are evaluated relative to the reference location. Positions can be set in pixels (default) or percentages.
Width Height	The width and height determine the size of the component. The width and height can be set in pixels (default) or percentages. Size keywords are also available to support special behavior.
Rendering Order	The order in which the component is rendered in the layer. A component with a larger order number will display above a component with a smaller order number. For components that support tab navigation (tabbing to the next editable component), the rendering order also determines the tabbing order. **NOTE:** On the Form Control Sheet, this setting is labeled as **Z-Index**.

Item	Description
Lock Layout	If enabled, the component size and position are locked and cannot be changed by dragging and dropping on the canvas. This optional setting is intended to protect against accidentally moving or resizing a component while working on the canvas.

NOTE: If a component belongs to a parent Panel component that is configured to use Flow layout behavior, then the position and order of that component is determined differently. For more information, see Auto-flow components in a panel.

Reference location

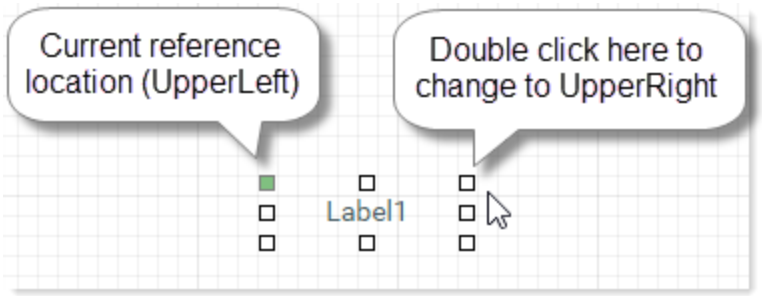
The **Reference Location** setting determines how the horizontal and vertical positions of the component are evaluated. By default, the reference location is set to **UpperLeft**, which means that the vertical y-position is evaluated from the top of the page and the horizontal x-position is evaluated from the left of the page. A component with y-position 0 and x-position 0 will display with the top left corner of the component in the top left corner of the page.

If desired, you can change this to another location, such as **LowerRight**. Now the y-position is evaluated from the bottom of the page and the x-position is evaluated from the right of the page. A component with y-position 0 and x-position 0 will display with the bottom right corner of the component in the bottom right corner of the page.

If a component belongs to a parent Panel component, then the reference location is evaluated within the context of that panel rather than the overall page. For example, if the child component has an UpperLeft reference location with x-position 0 and y-position 0, the component will display at the top left corner of the parent panel instead of the page. For more information, see Using panels to group and position components.

The reference location can be any of the following: **UpperLeft**, **UpperRight**, **LowerLeft**, **LowerRight**.

The reference location of a component can be changed by using the Form Designer. Locate the component on the canvas, then double-click the selection handle that corresponds to the desired reference location. For example, double-click the top right corner if you want to change the reference location to UpperRight. The selection handle for the current reference location is colored green.



When you change the reference location using the Form Designer, the x-position and y-position are automatically adjusted so that the component remains at its current location relative to the new reference location. Note that if you change the reference location in the Form Control Sheet directly, then the x-position and y-position remain the same, which means that the position of the component will be different relative to the new reference location.

Fixed and dynamic reference locations

The default reference location, UpperLeft, is a fixed reference location. However, if you select any other reference location, the reference location is now dynamic based on the current page size (or panel size). For example, if the reference location is LowerRight and the component is placed in the lower right corner, then the position of that component will adjust so that it remains in the lower right corner as the page is made taller or wider.

Generally speaking, dynamic reference locations should only be used when you want a component to display close to that location at all times. For example, LowerRight should only be used when you want the component to always display in the lower right corner, or close to it—such as to display a logo in the lower right corner of the page. LowerRight should not be used when you want the component to display in the top left quadrant.

This guideline has to do with how the browser determines when scroll bars are necessary. The browser can handle content that extends past the right side of the page or below the bottom edge of the page, by displaying scroll bars as needed. However, the browser cannot handle content that extends past the left side of the page or above the top edge of the page. These kinds of configurations can occur, if for example, the component's reference location is LowerRight but its y-position is set to something like 900px. If the page height is currently only 800px, the end result is that the component location would resolve off-screen past the top of the page, and therefore not display in the form.

This guideline is less important when components belong to a panel, because in that case the display of the components is constrained to the panel. Generally speaking, the position and size of the child components should not be configured such that they would extend out of the panel (and if they do, the display is then handled by the Overflow setting for the panel).

X and Y positions

The **X Position** setting determines the component's location along the horizontal axis, and the **Y Position** setting determines the component's location along the vertical axis. The starting point for either position is determined by the reference location. For example, if a component's reference position is UpperLeft, then an x-position of 50px is calculated from the left edge of the page. If a component's reference position is UpperRight, then an x-position of 50px is calculated from the right edge of the page.

If a component belongs to a parent Panel component, then the x-position and y-position are evaluated within the context of that panel rather than the overall page. For example, if a component's reference position is UpperLeft, then an x-position of 50px is calculated from the left edge of the panel instead of the page. For more information, see Using panels to group and position components.

The x-position and y-position can be specified as follows. The default measurement is pixels.

Position	Description
Pixels (example: 50px)	The component position is evaluated based on a fixed number of pixels relative to the specified Reference Location. If no units are specified on the position, the position will be interpreted as pixels. For example, if an entry is just 50, it will be interpreted as 50px.
Percentages (example: 50%)	The component position is evaluated based on a percentage of the current page size (or panel size), relative to the specified Reference Location. For example, if the x-position is 10% and the reference location is UpperLeft, the component will be placed at 10% of the current page width or panel width, from the left-hand side. Percentage positions are dynamic and will adjust as the page or panel is resized, so that the percentage is calculated against the current width or height. The minimum position is determined by the canvas size defined for the form. For example, if the canvas width is 400px and the x-position is 10%, then the minimum x-position of the component is 40px. (This minimum does not apply directly to child components of a panel, except that it potentially constrains the minimum position of the parent panel.) If you want to use percentages, you must first manually adjust the position setting to indicate the percentage unit. Once the unit has been changed, you can then move the component on the canvas and the new position will continue to be represented using a percentage. The same approach applies if you want to switch back to pixels from percentage.
<Blank>	The component position is inherited from the component style. If the style does not define a position, blank is treated the same as 0px. Certain component styles, such as the styles used by default for Wizard Panel components, define position properties for the component. In this case, the component-level position settings must be left blank in order for the style-level properties to apply. If you place a component on the canvas and then later decide you want to assign a style that includes position properties, you must manually clear the relevant position settings.

NOTE: When using dynamic position settings, **Scale to Fit** must be disabled (default behavior). If Scale to Fit is enabled, the dynamic settings may not resolve as expected.

Width and height

The **Width** and **Height** of a component can be specified as follows. The default measurement is pixels.

Position	Description
Pixels (example: 50px)	The component size is a fixed number of pixels. In this case, the size of the component will always be the same, regardless of the page size (or panel size). The exceptions are if **Scale to Fit** is enabled for the form, or when viewing the form on a tablet device (which uses its own scaling behavior to fit the tablet). If no units are specified on the size, the size will be interpreted as pixels. For example, if an entry is just 50, it will be interpreted as 50px.
Percentages (example: 50%)	The component size is calculated based on a percentage of the current page size. For example, if the component width is 50% and the page width is currently 1000px, then the component will render as 500px wide. If the component belongs to a parent Panel component, then the size is calculated based on a percentage of the current panel size. Percentage sizes are dynamic and will adjust as the page or panel is resized, so that the percentage is always calculated against the current width or height. The minimum size is determined by the canvas size defined for the form. For example, if the canvas width is 400px and the component width is 10%, then the minimum width of the component is 40px. (This minimum does not apply directly to child components of a panel, except that it potentially constrains the minimum size of the parent panel.) If you want to use percentages, you must first manually adjust the size setting to indicate the percentage unit. Once the unit been changed, you can then resize the component on the canvas and the new size will continue to be represented using a percentage. The same approach applies if you want to switch back to pixels from percentage. When using percentages for width or height, make sure that your percentage logically combines with the defined position of the component. The two settings together should not exceed 100% of the width or height of the page (or panel). For example, if your x-position is 30% and your width is 80%, then your component width is always going to extend past the edge of the page or panel.

Position	Description
Dock	The keyword `Dock` means that the component is sized to fill the remaining width or height of the page (or panel). For example, if the width is set to dock and the x-position of the component is 100px (with an UpperLeft reference location), then the component will start at 100px from the left side of the page or panel and extend to fill the remaining width of the page or panel. Docked sizes are dynamic and will adjust as the page or panel is resized, so that the component always fills the remaining width or height. Once a size has been specified as docked, the component cannot be resized away from its docked edge. If you want the component to no longer be docked, you must manually edit the size settings to specify a size in pixels or percentage instead.
Auto	The keyword `Auto` means that the component will automatically use an appropriate width or height. This option is primarily intended for components that have a built-in size. When you place a component like this on the canvas, the width or height will be set to Auto by default. Generally speaking, this option should only be used on components where it is automatically assigned (or where you have accidentally changed the component's setting and you want to change it back to Auto). If you manually assign Auto to a component that does not support it by default, the results may not be as you expect. The exception to this rule is the Label component. You can set either the width or the height to Auto, and the label will automatically adjust to fit the current text. This may be useful if the text shown in the label is dynamic. (Label components are set to Auto by default for height.)

NOTE: When using dynamic size settings, **Scale to Fit** must be disabled (default behavior). If Scale to Fit is enabled, the dynamic settings may not resolve as expected.

Rendering Order

The **Rendering Order** setting (also known as **Z-Index**) determines the order in which the component is rendered within the layer, relative to other components in the layer. If the component is a child of a parent Panel component, then the order is relative to the other components in the panel within the same layer.

By default, the rendering order determines whether a component is rendered above or below another component. A component with a larger order number will display above a component with a smaller number (for example, 10 displays above 2). The rendering order is set automatically when using the Form Designer to add a new component, and when using **Send to Back** and **Send to Front**. There is not much

purpose in editing the setting manually, as you would have to also know the rendering number of all other components in the layer to set the order appropriately.

However, there is one case where it is required to edit the rendering order manually, when using a Panel component that is set to **Flow** layout behavior. In this context, the rendering order determines the flow order of the child components. The flow order will be set automatically as you add new components to the panel, but if you want to edit the order after the fact then you must do it manually. For more information, see Auto-flow components in a panel.

Locking component position and size

Once you have a component positioned and sized the way you want it, you can optionally lock the component so that it cannot be accidentally moved or resized on the canvas while you are working with other components. To do this, enable the **Lock Layout** property.

When Lock Layout is enabled, the component cannot be moved or resized on the canvas. However, you can still adjust the position and size manually using the advanced component settings.

When using the Wizard Panel and Titled Panel components, these components are locked by default because they are intended to be used in a certain configuration.

Setting component position and size using styles

Component position and size can also be set using styles. For example, the **docked-to-container** style sets the x-position and y-position to 0px, and sets the width and height to dock. Axiom Software uses styles like this to apply preconfigured positions and sizes when you place certain components on the form canvas, such as the Wizard Panel or the Titled Panel.

When position or size is set by a style, the corresponding position or size property for the component must be blank. If the position or size property is not blank, then the component property will take priority over the style. This means:

- If you want to manually apply a style with position or size settings to a component, you should first apply the style, then clear out the relevant position or size settings for the component. You must do this even if you have just added the component to the canvas and not adjusted it at all, because Axiom Software writes the position and size settings to the component properties as soon as you drop the component on the canvas. (The exception is when you drop a preconfigured component on the canvas, such as the Wizard Panel.)
- If a component already uses a style with position or size settings, and you want to remove the style, you should first define a value for any blank position or size properties. Once all position and size properties have a value, you can remove the style. If you remove the style when blank properties exist, the component may not be positioned or sized as expected, which may cause issues when attempting to work with the component in the Form Designer.

Using multiple layers on the canvas

Each Axiom form canvas can have multiple layers. Layers can be used for two purposes:

- Within the Form Designer, layers can be used to assist in design, to show or hide groups of components that you want to work with.
- When the form is rendered, layers can be used to dynamically show and hide groups of components to the user.

Layers are one of the ways that you can design a single form that has multiple "pages" or "views". Each layer in the form can represent a different view that you want to dynamically show or hide. Axiom form viewers can then switch among the different layers by using an interactive component such as a combo box or radio buttons. Different layers can then become visible or hidden based on the user's selection. (Alternatively, Panel components can also be used to achieve this type of design.)

Conceptually this is similar to a tabbed interface, where users can click tabs to see different content, except that Axiom form designers can choose the means by which users change views, and can control what part of the form is affected by the change.

You can also use layers as a Form Designer tool only, to make it easier to show and hide certain groups of components. If your form has many components, this can make it easier to find and work with just the components you want.

NOTE: Using layers is the only way to hide components in the Form Designer. All components display in the Form Designer regardless of whether the **Visible** property for the component is enabled or disabled. This is done so that you can work with dynamically visible components on the canvas without having to clear or change the formulas that make them work.

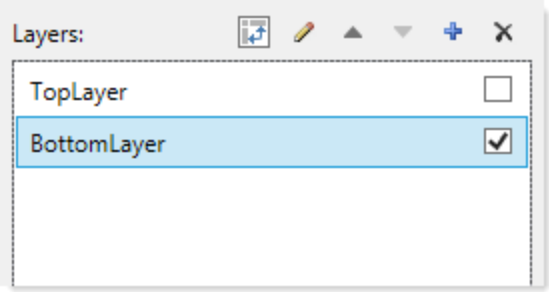
Layers have two properties that control their visibility:

- **Is Visible In Designer**: This property determines whether the layer is visible in the Form Designer. This property has no effect on whether the layer is visible or not in the rendered form. You can toggle layers visible or not by using the check boxes in the **Layer** section of the Form Designer.
- **Visible**: This property determines whether the layer is visible in the rendered form. This is the property that you would set up to dynamically change based on some other component in the form, so that form users can switch between viewing different layers.

By default, each Axiom form canvas has one layer named Layer1. If your form will only have one layer, then you can ignore this feature and leave the layer at its default settings. If you plan to use multiple layers, then it is a good idea to rename the default layer to something more descriptive.

Working with layers in the Form Designer

You can work with layers in the bottom right corner of the Form Designer. This area lists all layers defined for the Axiom form.



You can manage layers as follows:

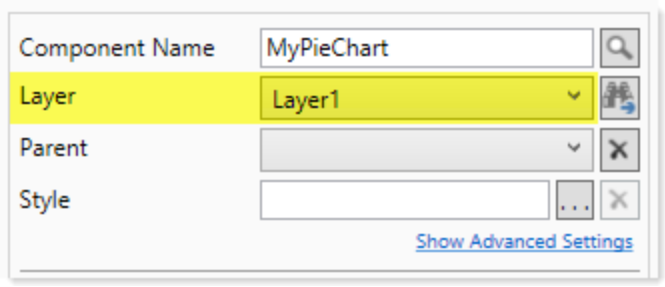
- **Adding a layer**: To add a layer, click the **Add layer** icon ✚. This adds a new layer to the list, named something like Layer2 (assuming you are adding a second layer). You should rename this layer to something more descriptive.
- **Renaming a layer**: To rename a layer, click the **Rename layer** icon (you can also right-click the layer). In the **Rename Layer** dialog, type the new layer name and then click **OK**.
- **Change layer order**: To move a layer up or down in the layer order, select the layer that you want to move and then click the **Move layer up** arrow or the **Move layer down** arrow. The layer at the top of the list is the top-most layer.

  The layer order is represented on the Form Control Sheet using the Z-index setting for each layer.
- **Deleting a layer**: To delete a layer, select the layer in the list and then click the **Delete layer** icon ✕ (you can also right-click the layer). You are prompted to confirm that you want to delete the layer. All components in the layer will also be deleted, so if you want to save any of the components you should first move them to a different layer.
- **Select all components**: To select all components in a layer, click the **Select all in layer** icon (you can also right-click the layer). All components that are assigned to that component become selected on the canvas.
- **Hiding or showing a layer**: To show a layer in the Form Designer, select the check box for that layer. To hide the layer, clear the check box. You can also hide and show layers using the Layer link at the top of the Form Assistant.

  This setting only affects whether layers display in the Form Designer and the Form Assistant for editing purposes; it does not affect whether the layer shows in the rendered Axiom form. The **Visible** property for the layer determines whether it is visible in the rendered form.

Assigning components to layers

When you drag a component onto the canvas, it must be assigned to a layer in the component properties.



When a new component is dragged and dropped onto the canvas, by default the component is assigned to whichever layer is currently active—meaning, whichever layer is currently selected in the **Layers** list. If there is only one layer, then that layer is selected by default.

NOTES:

- If you drag and drop a new component onto a Panel component, that component will be assigned to the same layer as the parent panel, regardless of the currently active layer for the canvas. This will cause the component's layer to become the active layer.
- If you duplicate a component, then the new component will inherit whichever layer was assigned to the original component.

Each component can only belong to one layer, although you can change the component's layer dynamically by setting up a formula on the **Layer** property for the component in the Form Control Sheet. However, if you find yourself wanting to assign a component to multiple layers, then you may want to reevaluate your Axiom form design. For example, you may need a "base" layer that is always visible, or you may need to change the visibility of some components at the component level instead of at the layer level.

Using the Form Designer, you can move components to a different layer as follows:

- To move an existing component to another layer: Select the component in the canvas, then click **Arrange > Move to layer**, then select the desired layer. This command is also available by right-clicking a component. The component will be assigned to the selected layer, in the same location on the canvas.
- To duplicate an existing component to another layer: Select the component on the canvas. Press the SHIFT key, then click **Duplicate**. This will create a copy of the existing component, at the same location on the canvas. You can then assign this duplicate component to a new layer (by editing the component properties, or by using **Arrange > Move to layer**).

Configuring layer visibility in the Axiom form

The Layer check boxes in the editor only determine whether a particular layer is visible in the editor, so that you can work with different layers as needed. To determine whether a layer is visible in a rendered Axiom form, you must use the layer's **Visible** property.

By default, all layers are visible. If you want users to be able to dynamically show and hide different layers, then you should configure the layer settings so that certain layers are visible or not based on an interactive component, such as a combo box or radio buttons.

For example, imagine you have an Axiom form with two different "views" (layers): a Summary view and a Detail view. You want the Axiom form users to be able to choose which view they want to look at. You can set up a combo box where the users can select either Summary or Detail. Then you can configure the Visible property for each layer to point to the combo box selection, and either hide or show the layer depending on what is selected.

The Visible property for a layer can be edited on the Form Control Sheet. Layer settings are listed in the Components section of the control sheet, by layer name. You can also go directly to a layer's properties by clicking the **Show layer definition** button  next to the Layer setting for any component.

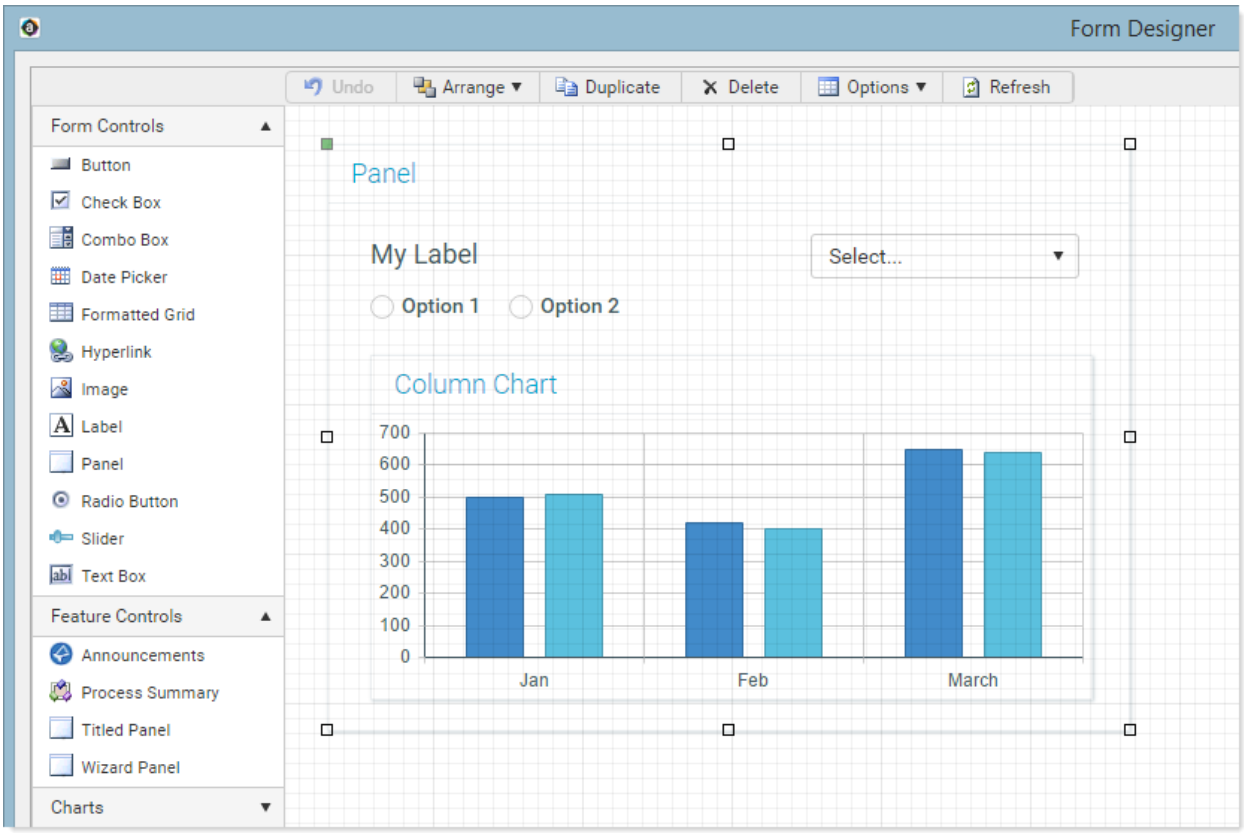
If more than one layer is visible within an Axiom form, the components of those layers will display in layer order first, then within each layer by component order (rendering order). The layer at the top of the Layer list is the top-most layer. This means that the components of one layer can cover some or all of the components of another layer if both layers are visible at the same time and the components overlap.

Using panels to group and position components

You can use Panel components on the form canvas to create groups of components and control their display relative to the panel. When using a panel:

- You define the overall area devoted to the panel, as well as any panel properties (such as a panel title or border).
- You assign one or more "child" components to the panel, by placing these components within the panel boundaries.

Once components have been assigned to the panel, the panel and all of its child components move as a group on the form canvas. Additionally, the child components can now be sized and positioned relative to the panel instead of relative to the overall canvas.



Example Panel component with child components

This screenshot shows an example Panel component in the Form Designer. This panel has several child components such as a label, radio buttons, combo box, and a chart. As the panel is moved on the canvas, the child components move with it.

NOTE: This section discusses the behavior of Panel components and their child components when using the default **Child Layout** option of **Positioned**. Panel components also support an alternate option named **Flow**, which allows child components to automatically flow across and down a page based on a set order. This alternate layout behavior is intended for home pages and dashboards where the content may not need to be positioned absolutely on a page. For more information, see Auto-flow components in a panel.

Creating a panel

The first step to using a panel is to drag and drop a Panel component on the canvas, and then size the panel to match the area that you want to control. You can resize the panel later, but you want to set an initial size that is large enough to drag and drop all of the desired child components within the panel.

In some cases, you may want to size the panel so that it fills the entire page. You can use the **docked-to-container** style for this purpose. Assign the style to the Panel component, then click **Show Advanced**

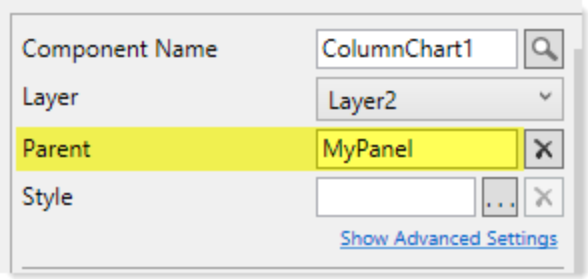
Settings and clear the position and size settings for the component so that they will be inherited from the style.

The Panel component does not have many component settings. One key decision to make is whether you want the panel shape to display as a design element on the form (for example, by giving the panel a border and/or a title), or whether the panel should be an "invisible" component used only for the purposes of controlling the child components. If a panel has no visible title bar, border, or background color, then the form viewer will not be aware of the presence of the panel. For more information about the panel settings, see Panel component.

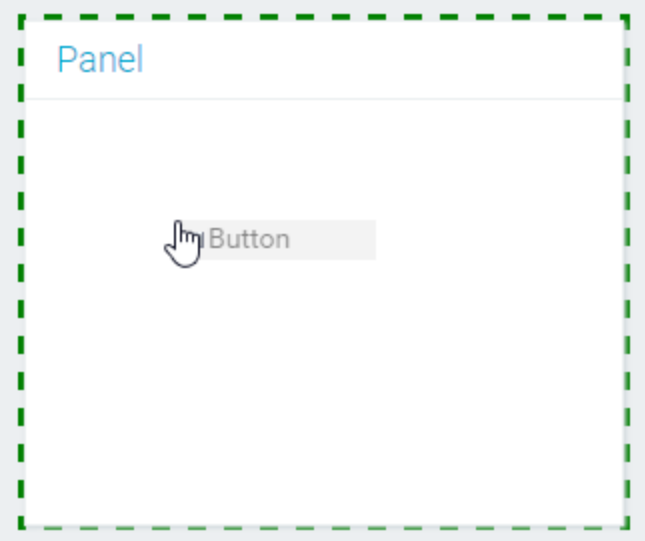
NOTE: The Wizard Panel component is also a Panel component, but it is specifically intended to support creation of wizard-style interfaces. By default, the Wizard Panel is sized to fill the full window and contains built-in wizard features and formatting. For more information, see Wizard Panel component .

Assigning child components to a panel

The primary way to assign child components to a panel is to use the Form Designer. When you drag and drop a component in a panel, the component is automatically assigned to that panel, by writing the name of the panel to the **Parent** property for the component. Any component can be assigned to a panel, including another panel (nested panels).



When dragging and dropping the child component to the panel, the top left-hand corner of the child component must be within the panel. Any other part of the child component can extend outside of the panel boundaries, as long as the top left-hand corner is within the panel (though in most cases it is desired for the component to be fully enclosed within the panel). When you are moving a component, you can tell that it will become assigned to a panel (or keep its existing assignment), because the parent panel is highlighted using a dashed border.



Example Button component will be assigned to highlighted panel

To remove a child component from a panel, drag and drop the component so that the top left-hand corner of the component is outside of the panel boundaries. The parent assignment will be automatically cleared (or updated to a different panel, if applicable).

NOTE: It is not recommended to manually edit the Parent property to assign a component to a panel or to remove an existing assignment, as this manual edit does not adjust the location settings for the component. Instead, the component's current location settings are left unchanged and are now interpreted as relative to the panel (or in the case of removing a parent assignment, as relative to the overall canvas). See the following section for more information about how the location of child components is determined. This mismatch of child component location and parent assignment can also result in confusing behavior in the Form Designer.

Child component position and size within the panel

Once a component has been assigned to a panel, its position and size in the form is now determined relative to the panel area instead of the overall canvas.

For example, if a component with no panel assignment has an x-position of 30px and a y-position of 30px, this means that this component is positioned 30 pixels from the top of the canvas, and 30 pixels from the left edge of the canvas (assuming the default reference location of UpperLeft). However, if the component is assigned to a panel, then the same settings mean that the component is positioned 30 pixels from the top of the panel and 30 pixels from the left edge of the panel, wherever that panel is located on the canvas. This keeps the position of the child components fixed within the panel as the panel is moved around the canvas.

Additionally, if a child component uses a dynamic position or size option, such as percentages or dock, then the position or size of the child component is now calculated against the panel, not the page. For

example, if a child component is set to 30% width, that component will be sized to 30% of the panel width, not the page width.

Child components only display in the rendered form if the panel is visible in the form. If the panel is hidden (by setting **Visible** to **Off** on the panel or on the layer it is assigned to), then the child components are also hidden, because there is no available reference to determine their position.

Make sure to consider the **Overflow** property when positioning and sizing panels and their child components. The overflow property determines what happens if a child component exceeds the borders of the panel component. Because only the top left-hand corner of the child component must placed within the panel boundaries in order to remain assigned to the panel, the rest of the component may extend beyond the panel boundaries. Even if the child component is fully contained within the panel when it is originally assigned, this may change due to situations such as the following:

- The parent panel is dynamically sized using a percentage, but the child components have fixed sizes using pixels. In this situation it is possible that the panel may render smaller than the fixed sizes of the child components, depending on the size of the page.
- You manually change the size of a child component or the parent panel, such that the child component's width or height now extends past the panel boundaries.
- You manually change the position of a child component such that it now extends past the panel boundaries.

By default, Panel components are configured so that overflow is visible in the form. If desired, you can change this so that overflow is hidden, or so that the panel scrolls to provide access to the overflow.

Working with panels in the Form Designer

Note the following behavior when working with panels in the Form Designer:

- When you drag and drop a panel to another location, or otherwise cause the panel to adjust location—such as by using the align or distribute options—then all child components in the panel move with it. As discussed in the previous section, this because the child components are positioned relative to the panel.
- Duplicating a panel only duplicates the Panel component, it does not duplicate the child components in the panel.
- Deleting a panel deletes the panel and all child components. If you do not want the child components to be deleted, you must move them out of the panel first. If you have already deleted the panel, you can use **Undo** to restore the panel and its child components (as long as you have not exited the Form Designer after deleting the panel).
- Child components within the same panel can be aligned and distributed with each other, but not with components outside of the panel.

- Child components can be moved to the front or back relative to other components within the panel, but they cannot move "underneath" the parent panel. The parent panel is always at the bottom relative to the child components in the panel. If external components overlap the panel (but do not belong to the panel), then the panel and its child components are treated as a single unit when moving the external component to the front or the back.

Using layers with panels

Panels and child components can belong to layers like any other component. Typically the panel and its children will all belong to the same layer, but they do not have to. You might use layers in this situation simply as a Form Designer tool, to control which components display in the Form Designer at any one time. You can also use layers to control visibility in the rendered form, if controlling visibility at the panel level is not sufficient.

When using layers with child components of a panel, keep in mind that child components will only be visible if the parent panel is visible. For example, imagine that the panel belongs to a layer named Panel and a child component belongs to a layer named Child. The visibility of these components based on layer visibility is as follows:

- If both layers (Panel and Child) are visible, then the panel and the child are visible.
- If layer Panel is visible but layer Child is hidden, then the panel shows but the child is hidden.
- If layer Panel is hidden but layer Child is visible, then both the panel and the child are hidden. The child component cannot be visible if the parent panel is hidden, even if the child component's layer is visible.

While working in the Form Designer, the following layer behavior applies to panels:

- When you drag and drop a new component into a panel, the child component will be automatically assigned to the same layer as the parent panel. However, if you drag and drop an existing component into a panel, the child component will retain its current layer. In this case you must manually change the layer for the child component if you want it to have the same layer as the panel.
- If a panel is not currently visible in the Form Designer (due to hiding it via a layer), then you cannot drag and drop components into that panel. The hidden panel is ignored for assignment purposes. Additionally, all of the hidden panel's child components will also be hidden. This allows you to work with multiple stacked panels, by assigning each one to a layer and then showing one layer at a time.
- If two visible panels overlap and the panels belong to different layers, the layer order determines which panel a component is assigned to when it is dragged and dropped on the overlapping area.
- If you change the layer for a parent panel, this does not change the layer of the child components. You must change the layer for each individual component that you want to move to the same layer as the parent panel.

Using stacked panels

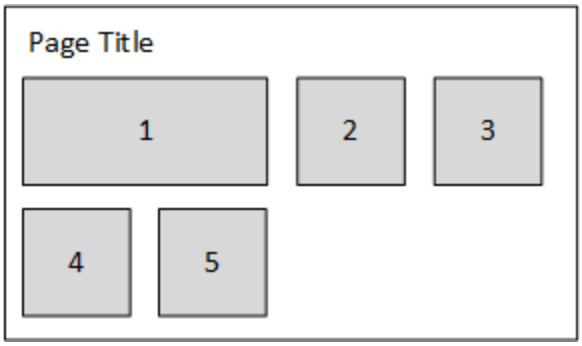
Panels can be used simply for the purpose of organizing and controlling groups of components, or they can also be used to create forms with several different "pages" or "views". Your form could have several "stacked" panels that you show and hide based on the user's current selection from another component, such as radio buttons or a combo box (or by using a Wizard Panel).

When using stacked panels, you must also use layers and assign the panels to layers, so that you can show and hide layers in the Form Designer and work with only the panel or panels that you want to. Otherwise you would need to continually use Send to Back / Bring to Front to move the panel you want to work with to the front. You cannot simply select the panel that you want to work with, because if that panel is actually "behind" the other stacked panels (based on rendering order), then any child components moved into or around the selected panel will actually be assigned to the panel at the front of the stack. Instead you must either move the selected panel to the front of the stack, or hide the other stacked panels in the Form Designer using layers.

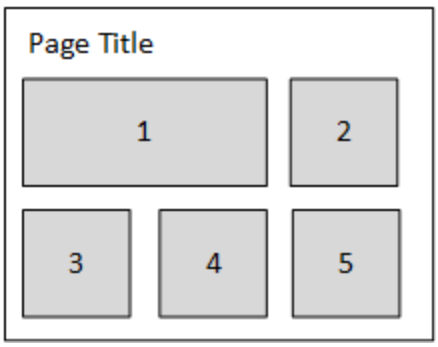
Auto-flow components in a panel

Using Panel components, you can design a form where child components are not absolutely positioned. Instead, the child components will automatically flow across and down the panel as needed, based on a defined order and depending on the current page size. This feature is primarily intended for home pages and dashboards where you may have several "blocks" of content that you want to present in a certain order, but you want these blocks to flow to fit the page dynamically.

For example, imagine that you have five main content blocks. These "blocks" could be components such as the Announcements component, the Process Summary component, one or more Formatted Grid components, and one or more chart components of various types. You want these content blocks to automatically flow to fit the page, so that on a wide screen the components might look like this:



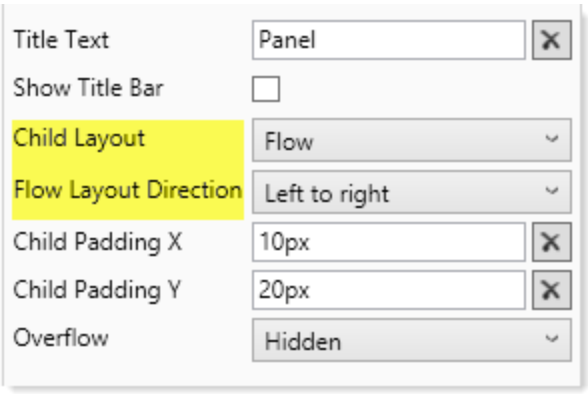
But on a smaller screen or a resized window, the components might look like this:



To do this, you place a Panel component on the form canvas that defines the area where you want child components to flow. All child components assigned to the panel will automatically flow to fill the panel in a designated order. Components start by flowing from left to right, from the top left corner of the panel. The first component that would extend past the current panel width is then dropped down to start the next row of components, and components flow left to right again.

To configure a Panel component for flow layout behavior:

- Place a Panel component on the form, and position and size it as desired. For more information on design considerations for the parent panel, see the following section.
- Set the **Child Layout** property of the panel to **Flow**. This means that the child components of the panel will now auto-flow across and down the panel, instead of using the component properties for x-position and y-position to determine absolute positioning.
- Set the **Flow Layout Direction** to the desired flow direction. By default, child components flow from left to right. However, you can optionally change this so that child components flow from right to left.



Panel properties configured for flow layout

Positioning and sizing the parent Panel component

The Panel component defines the area where you want components to flow. For example, you might want the panel to take up the whole page (so that the flow behavior applies to the whole page), or you may want to dedicate the top area of the page to fixed titles and other content, and use the panel to fill the remaining area of the page.

Note the following when positioning and sizing the panel component:

- In most cases, you will want a page title at the top of the page that is not part of the panel flow. Therefore you should set the **Y Position** of the panel to something like 100px, so that you can place a fixed position label at the top of the page (and other components as needed, such as an image). The panel and its flow behavior can start underneath this fixed area.
- Set the **X Position** of the panel to the amount of padding you want between the left side of the page and the start of the panel contents. For example, you might set this to 10px.
- If you want the panel to fill the remaining width and/or height of the page, then set the Width and/or Height to `dock`. The child padding settings will ensure that panel contents do not bump directly against the right edge and bottom edge of the page. For more information, see Setting the padding between child components. If you want the panel to fill a designated area of the page, set the Width and/or Height to a specific size (pixels or percentage).
- Make sure to configure the **Overflow** property as appropriate for the desired behavior. In the majority of cases you should set this to **Visible** or **Scroll** so that the user can see all child components regardless of the current panel size. Visible will cause the page to scroll as needed, and Scroll will scroll the panel as needed. Using **Hidden** with any panel configuration may result in child components flowing off the panel and therefore not visible to users.

To edit the size and position properties for the Panel component, click **Show Advanced Settings** in the Form Designer or Form Assistant.

Once you have the panel positioned and sized the way you want it, it is a good idea to enable **Lock Layout** so that you do not accidentally move or resize it when working in the Form Designer. You can still edit the position and size settings manually while the layout is locked.

Placing child components on the panel

Once the Panel component is configured for flow layout behavior, you can now drag and drop child components into the panel, in the order you want them to flow. The first component added to the panel becomes the first component in the flow order, and so on. You can change the order later, but if you already know the desired order it is easiest to set it when you are initially adding components to the panel.

Note the following behavior when adding child components to a flow panel:

- The **X Position** and **Y Position** properties are blank by default for child components of a flow panel. These settings are irrelevant due to the flow behavior.
- By default, **Lock Layout** is enabled for child components, so that you cannot move or resize them on the canvas. This is done to emphasize that the component position is controlled by the flow order—moving a component on the canvas has no effect as long as it belongs to a flow panel.
- If you want to resize a component, you can click **Show Advanced Settings** and manually enter the desired width and height, or you can disable **Lock Layout** and then resize the component on the canvas. When you are done resizing the component, it is recommended to re-enable Lock Layout.

  NOTE: The act of resizing a component on the canvas may cause the x-position and y-position to become populated with values. These values will be ignored as long as the component belongs to the parent panel. You may want to manually clear these values so that they do not cause any confusion.

- The **Rendering Order** of the component is automatically set to the next highest number of all components within the layer. In this context, the rendering order is the flow order. The child component with the smallest number renders first in the flow, and the component with the largest number renders last in the flow.

  As you drag and drop components into the panel, they will display in flow order within the Form Designer. This is the same flow that will be used in the browser when the form is rendered.

If you need to change the flow order after placing components on the canvas, you can do this by manually editing the **Rendering Order** in the advanced component settings. The flow order *cannot* be changed by moving components on the canvas—for example, if you clear Lock Layout and then drag child component 6 to a spot before child component 4, this will not change their order (and component 6 will automatically snap back to its flow position when you are done). You must manually edit the rendering order for all affected components if you want to change the flow order. After making these edits, you must refresh the Form Designer to see the effects of the changes.

The specific numbers in the rendering order do not matter; the only thing that matters is how each number relates to the other child components in the panel (smaller or larger). Additionally, you should not expect the number for the first child component to be 1, because the numbers are auto-generated based on all components in the layer (including the parent panel itself). When reordering components, you can skip numbers to create more leeway for future reordering.

All child components in a flow panel should have unique rendering order numbers. If two child components have the same rendering order number, then Axiom Software will determine their order. The numbers only need to be unique within the context of the child components of that panel. Other components that do not belong to the panel or that belong to different panels can duplicate rendering order numbers.

NOTES:

- When dragging and dropping components into the panel, make sure that the top left-hand corner of each component is within the panel, so that the panel is automatically assigned as the **Parent** of the component. Also, remember that you do not need to precisely position the child components while you are dragging and dropping them in the panel, because the child components will be automatically positioned due to the flow. For example, if you are dragging and dropping the third child component into the panel, there is no need to try and position the component "after" the second component.
- Although it is possible to place components on a **Fixed Position** panel and then later change the panel to **Flow**, the components will probably not end up in the order you want and you will need to manually reorder them. It is best to set the panel to use the flow layout behavior before adding child components.
- If you added a component to the panel by mistake and you want to move it out of the panel, you can clear Lock Layout in the advanced component settings and then drag it outside of the panel. This will clear the Parent setting so that the component no longer belongs to the panel, and you can now position the component as normal.
- If you duplicate a component in a flow panel, the rendering order number of the new component will be the same as the original component. You should manually edit this number as appropriate, depending on where you want the new component to be placed in the flow order.

Setting the padding between child components

When a Panel component is set to flow layout behavior, two new settings are exposed for the panel, to control the padding between child components:

- **Child Padding X** defines the horizontal spacing between components. It is applied to the right side of each component when using the default left-to-right flow, and to the left side of each component when using right-to-left flow.
- **Child Padding Y** defines the vertical spacing between components. It is applied to the bottom of each component.

For example, you might want to set the x-padding to 10px so that there is 10px of space between each component within a row, and set the y-padding to 20px so that there is 20px of space between each row of components. The padding can be set in pixels (default) or percentages.

The padding is incorporated into the child component's overall size. For example, if a component is set to 400px width and the x-padding is 10px, this means that the component contents take up 390px so that there is 10px left over for the padding.

The padding applies to all child components in the panel, including the last component in a row and the bottom row of components, so that the panel contents do not bump up directly against the right edge or bottom edge of the panel.

No padding is applied to the top of child components, or to the starting side of child components (for example, the left side when flowing left-to-right). This means that the parent Panel component should not be placed directly against the left side of the page or the top of the page, or else child components will bump directly against the left and top of the page. Instead, you should set the x-position and y-position of the Panel component so that it provides the necessary space between the left and top of the page. For example, you might set the x-position of the Panel component to the same amount as the x-padding for the child components, so that there is a consistent horizontal padding for all components in the row. (If you have changed the flow direction so that it flows from right-to-left, the same principle applies but now it applies to the right side.)

Using panels within a flow panel

You may have multiple components that you want to keep together as a "block" within the flow layout. To do this, you can add a Panel component as a child of the parent flow panel, and then add the components to the child panel. The child panel will have a flow order like all of the other child components of the flow panel, and will adjust position as the flow panel is sized larger or smaller. But the components within the child panel can be absolutely positioned within that panel, so that they maintain their positions as the child panel adjusts within the flow.

For example, you may have a label and three bullet charts that you want to display as a block within the flow layout. You can:

- Add a Panel component to the flow panel, and make sure its rendering order is set appropriately for where you want this content positioned within the flow. The **Child Layout** for this panel should be left at the default of **Positioned**. Adjust the size of the panel as needed to hold the contents that you want to display within this "block."
- Add the label and the bullet charts to this child panel, and size and position them as required. For example, the label may go at the top of the panel, followed by three stacked bullet charts. When you drag and drop components into this child panel, make sure they are fully within the child panel. If a component extends outside of the child panel, then it will not be assigned to that panel and will instead be assigned to the parent flow panel.

As this child panel flows within its parent panel, the individual components belonging to the child panel will stay together and maintain their positions within the child panel.

Component flow and canvas size

The canvas size of the form—specifically, the canvas width—determines the minimum number of child components that will display on a row. For example, if the canvas width is 800px, and the first three components are 400px, 300px, and 300px wide respectively, then the first two components will always display on the first row of the panel because 800px is the minimum width. If the page is sized narrower than that, then a horizontal scroll bar will result instead of pushing the second component to the next row.

You should consider the display of the form and adjust the canvas size as appropriate when setting up a flow panel. By default, the canvas width is 400px. Unless you have several narrow components at the

start of the flow order, this likely means the form will allow resizing to the point of showing only one component per row. In many cases this may be an undesirable result. If so, you should set the canvas width to a size that accommodates the minimum number of components that you want to display on a row.

To change the canvas width, click **Edit Form Properties** in the Form Assistant or Form Designer, and then edit the first number under **Canvas Size**. For more information, see Defining the canvas size of an Axiom form.

NOTE: **Scale to Fit** cannot be used with flow panels. Generally speaking, scale-to-fit behavior is not compatible with any dynamic size and position option.

Defining the canvas size of an Axiom form

The canvas size of an Axiom form defines the minimum width and height of the form web page when it is rendered. By default, the canvas size is 400 x 400 pixels. However, the effective canvas size will adjust automatically based on the fixed contents of your form.

The minimum width and height apply when the form contains any components with dynamic position or size settings. For example, you may have a component that is sized at 50% of the page width. In this case, the canvas width determines the smallest size the component can be. If the canvas width is 800px, then the minimum width of the component is 400px. The component will not get smaller than that, even if the browser window is smaller than the canvas width (in this case, a scroll bar will result). For more information on using dynamic size and position settings for components, see Controlling component position and size.

The canvas size only affects the minimum width and height of the form. It does not define a maximum size. If the browser window is larger than the canvas size, then dynamic components will adjust to the larger size.

In the majority of cases, you should not need to adjust the canvas size. The built-in sizing behavior is intended to meet the needs of most forms without requiring manual intervention by the form designer.

NOTES:

- For Axiom forms that are being used as custom dialogs in the Desktop Client, the canvas size defines the size of the dialog. For more information on this use case, see Custom Dialogs and Task Panes in the Desktop Client.
- The default canvas size behavior described in this topic does not apply if **Scale to Fit** is enabled for the form. See Using scale to fit (legacy form sizing).

Defining the default canvas size

The default canvas size of 400 x 400 is determined by the form-level skin. Currently, all skins delivered with the Axiom Software platform set a canvas size of 400 x 400.

If desired, you can set a different default canvas size on a per form basis. To do this:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, edit the **Canvas size** by entering new values for width and/or height. The width is the first value and height is the second value.

TIP: You can also change the canvas size by editing the **Width** and **Height** fields at the top of the Form Control Sheet.

The default canvas size will be overridden automatically by any fixed components that exceed the default canvas size. In that case, the default canvas size is ignored and the canvas size defined by the form contents is used instead to determine the minimum width and height of the rendered form.

How form contents affect canvas size

The canvas size is not a static property; it will automatically adjust based on the fixed contents of your form. "Fixed contents" refers to any components with the following properties:

- Reference Location: UpperLeft
- X and Y Position: Pixels
- Width and Height: Pixels

If a fixed component exceeds the default canvas width or canvas height, then the effective canvas width or height automatically adjusts to include the fixed component. Essentially, by placing a fixed component outside of the boundaries of the default canvas size, you are implicitly overriding that canvas size.

For example, imagine that the default canvas width is 400px. Then, you add a component that is 200px wide and position it 300px from the left edge. This component exceeds the default canvas width by 100px. As a result, the canvas width is now effectively 500px. When the form is rendered, any components with dynamic position or size will use 500px as the minimum width of the form.

In most cases, you do not need to be aware that you are placing components outside of the default canvas size. The goal of the canvas size behavior is to eliminate the need to think about and manually adjust the canvas size.

However, in some cases you may have a specific "target size" for the form, and you need to be able to see and account for these dimensions while designing the form canvas. In that case, you can toggle the canvas size to be visible in the Form Designer. To do this, click **Options > Show form canvas area**. When this option is enabled, the canvas size is shaded on the canvas, so that you can tell if you have components that exceed the canvas size.

Scale to Fit and canvas size

As of version 2016.1, the **Scale to Fit** option is considered a legacy option that should not be used for new forms going forward. Many new features, such as the dynamic size and position options for components, are not intended to be used with scale-to-fit.

The only reason to use scale-to-fit is if you are an upgrading customer who is not yet ready to migrate your forms to the new features. In this case, you may still need to create new forms that fit your existing environment. For more information, see Using scale to fit (legacy form sizing).

Using scale to fit (legacy form sizing)

The **Scale to Fit** option for Axiom forms causes all form contents to automatically scale larger or smaller to fit the size of the current window. As of version 2016.1, scale-to-fit is considered a legacy option that should not be used for new forms going forward. Instead, the dynamic size and position options for components should be used.

Going forward, the only reason to use scale-to-fit is if you are an upgrading customer who is not yet ready to migrate your forms to the new features. In this case, you may still need to create new forms that fit your existing environment. If all or most of your other forms use scale-to-fit, then you may want to enable it for your new forms.

Creating new forms for use with Scale to Fit

If you need to create new forms for use in your legacy environment, then by default these new forms will not have a defined canvas size, and will not have scale-to-fit enabled. Therefore, when creating a new form, you must do the following if you want to use scale-to-fit:

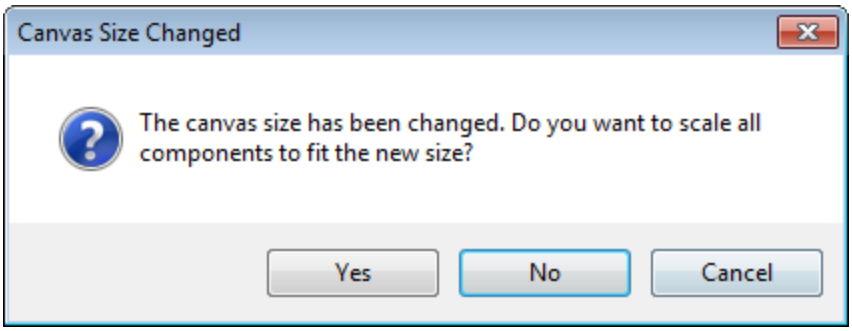
- Define the **Canvas Size** (width x height). For reference, the legacy default canvas size for use with scale-to-fit used to be 1200 x 800. You should define a canvas size that encompasses all of the contents of your form. If needed, you can show the canvas size in the Form Designer by going to **Options > Show form canvas area**.
- Enable **Scale to Fit**.

Both of these settings can be edited in the **Form Properties** dialog. From the Form Designer or Form Assistant, click **Edit Form Properties** to access this dialog.

Defining a canvas size is necessary when using scale-to-fit, because the default 400 x 400 canvas size is likely too small for scaling. The default canvas size behavior does not apply when using scale-to-fit.

Working with canvas size and scale-to-fit

If you change the canvas size when **Scale to Fit** is enabled, you have the choice of whether or not to automatically scale any existing components on the canvas to fit the new size.



- In general, you should select **Yes** if you have used the full area of the canvas and now you want to resize the entire form and its contents. This will ensure that the form components fit on the new canvas area in the same proportions that they do currently, without any form components being "orphaned" outside the canvas area.
- You should select **No** if your canvas currently has a lot of unused white space, and the reason you are resizing the form is to get rid of this unnecessary white space.

NOTE: The ability to automatically adjust form components for the new canvas size only applies if you change the canvas size using the **Form Properties** dialog. If you edit the canvas size on the Form Control Sheet directly, then only the canvas size will change and individual components will not be adjusted.

Disabling scale-to-fit for existing forms

Going forward, we recommend disabling scale-to-fit for all forms, regardless of whether you are ready to adopt other new form features such as the Web Client container or dynamic size and position for components.

If you decide to disable scale-to-fit for an existing form, you should also consider clearing the canvas size for the form, so that you can use the new behavior for canvas size. For more information, see Defining the canvas size of an Axiom form.

Setting the background color or image for an Axiom form

By default, the background color of the form web page is determined by the skin assigned to the form. If desired, you can use the following form-level settings to change this display:

- Background Color
- Background Image

The background color and image are not mutually exclusive. Any area not covered by the background image will display using the background color (whether it is explicitly set or determined by the skin).

It is also possible to configure a Panel component so that it extends to fill the entire page, and therefore effectively defines the background color of the form (for example, when using the Titled Panel

component). In this case the background settings for the form still apply, but since the panel covers the entire page, they will not be seen (unless the panel is translucent).

Setting the background color

You can assign a background color to a form. The background color fills the entire tab or window where the form is being displayed.

To specify a background color for a form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, edit the **Background Color** by selecting a color.

   Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color.

TIP: You can also change the background color by editing the **Background Color** field at the top of the Form Control Sheet. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

Setting the background image

You can display an image in the background of an Axiom form.

To specify a background image for a form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, edit the following settings:

Item	Description
Background Image	Specifies the image file to use for the background image. Click the **[...]** button to browse to the image within the Reports Library. If the image is not already saved in the Reports Library, you can right-click a folder and select **Import** to import the image (if you have the appropriate rights to do so). The image must be in PNG or JPG format. **NOTES:** • Users must have permission to the image file in order to see it rendered in the form. It is recommended to create a dedicated Images folder in the Reports Library and store all images in this location. You can grant access to this folder using the Everyone role, or you can create subfolders and grant access to users and roles as needed. • The next time you open this file after saving, the path to the image will be automatically converted into a system-managed document shortcut (you can tell the difference by the presence of a `_tid` parameter on the end of the shortcut). This is to make the file reference "repairable" in cases where the file is renamed or moved. Note that if the path is a result of a formula instead of directly within the cell, then the conversion will not occur and the file reference will not be repairable.
Image Repeat	The repeat behavior of the image. Select one of the following: • **Both** (default): The image repeats in a tiled pattern both horizontally and vertically, covering the entire form background. • **Horizontal**: The image repeats in a tiled pattern horizontally, starting at the top left corner of the form and extending all the way across. • **Vertical**: The image repeats in a tiled pattern vertically, starting at the top left corner of the form and extending all the way down. • **None**: The image is not repeated. The image displays in the top left corner of the form.

TIP: You can also change the background image by editing the **Background Image Path** and **Background Image Repeat** fields at the top of the Form Control Sheet. If you are specifying the path manually, use the full path in the Axiom file system.

Controlling the Axiom form appearance with skins and styles

The appearance of an Axiom form is controlled by the following settings:

- **Skin**: The skin sets the overall look and feel of the form. Skins determine the default colors used in the form, as well as other styling aspects such as flat or 3-D design treatments.
- **Styles**: Styles can be applied at the component level to further refine the formatting for that individual component. For example, you might apply a title style to a Label component that is being used as a title in the form, so that the text displays in a larger font, or in bold font (or both).

Many form components also allow you to define specific formatting properties—such as font size and border width— within the component settings. If defined at the component level, these settings override formatting inherited from the skin and style.

Skins and styles are intended to help you create forms that look polished and consistent within each individual form, and across all forms created by your organization. For example, using the same skin across all forms helps ensure that all your forms have a similar look and feel, and using styles helps ensure formatting consistency for individual components.

Legacy themes

If your form uses any skin other than the default Axiom2018 skin, there is another layer of formatting known as the *theme*. The theme sets the overall formatting for the components in the form, and determines which styles are available for use in the form. Generally speaking, the theme formatting is designed to fit particular use cases for forms. For example, some themes are intended for display of reporting data, while other themes are intended for input forms.

Themes have been deprecated going forward because user feedback indicated that it was difficult to tell which theme to use and difficult to manage different styles for different themes, particularly with Formatted Grid components. In response to this feedback we have eliminated the themes and created a new style structure that directly defines formatting properties rather than using semantic styles. However, this new approach only applies to the new Axiom2018 skin. When working with older forms, you must continue to use themes until you migrate the form to the new skin.

For more information on using themes, see Setting the theme for an Axiom form (deprecated).

Setting the skin for an Axiom form

The skin sets the surface look and feel of an Axiom form. The goal of the skin is to promote a consistent overall appearance across all of the forms used at your organization, by defining certain high-level design elements. The skin also determines which styles are available for use in the form.

Generally speaking, Axiom Software is designed to work best with a single primary skin. The current primary skin is Axiom2018. This skin contains the latest Axiom Software design elements and styles. By

default, all newly created forms use this skin. All other skins are primarily intended to support backward-compatibility.

The Axiom2018 skin and the legacy skins are not designed to be interchangeable. Generally speaking, you cannot change the skin from Axiom2018 to a legacy skin (or vice versa) and have the form continue to display as expected. The primary differences between skins are the row and column styles used by Formatted Grid components, but other component styles may not be recognized when the skin is changed. Unless the form is very simple, changing the skin requires additional manual adjustments so that the form uses the styles expected by skin. Also, newer components may not have been optimized for use with legacy skins. For more information on converting an existing legacy form to use the new Axiom2018 skin, see Migrating an existing form to use the Axiom2018 skin.

Setting the system default skin

Each system has a default skin that is applied to newly created forms. The system default skin is specified using the **WebClientSkin** property in the system configuration settings. By default, this property is set to **Axiom2018**, which means that when a new form is created, the **Skin** property for the form is automatically set to **Axiom2018**.

If desired, you can edit this property to specify any skin that is available to be selected for an Axiom form. However, this is primarily intended for backward-compatibility, so that systems that were created using an older skin can continue to use that skin until they are ready to migrate their forms to the current Axiom default skin.

For more information on editing the system configuration settings, see the *System Administration Guide*.

Setting the form-level skin

When a new form is created, the **Skin** property for the form is set to the system default skin. This means that all new forms use the system default skin to start, though this can be changed on a per form basis as needed. By default this skin is **Axiom2018**, though your system may be configured to use a different skin for backward-compatibility reasons.

Under most circumstances, it is not necessary to change the skin for an Axiom form. However, you may need to change the skin if you are migrating an existing form from using a legacy skin to the latest Axiom default skin.

To set the skin for an Axiom form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, select the desired skin from the **Skin** drop-down list.

TIP: You can also change the skin by editing the **Skin** field at the top of the Form Control Sheet. In this case you must type in the desired skin name.

The skin is loaded when the form is opened and cannot be changed dynamically during the current session.

Migrating an existing form to use the Axiom2018 skin

If you have existing forms that you want to migrate to the Axiom2018 skin, some manual adjustments will likely be necessary after changing the skin.

At minimum, you will need to do the following after changing the skin:

- If the form uses thematic Formatted Grid components, you must update the row and column styles in the grids to use the new styles supported by the Axiom2018 skin. These new styles allow you to directly apply certain formatting features, to more precisely control the formatting in the grid. For more information, see Using row and column styles in a thematic grid. The previously assigned row and column styles will not be recognized by the Axiom2018 skin. Until the styles are updated, the grid will display as if it has no formatting.
- If the form uses h1-h5 styles for Label components, these styles are not supported by the Axiom2018 skin. You must update the labels to instead use the new styles for font size and font color.
- If the form uses a Wizard Panel component, you must change the style on the grid or panels that provide the wizard contents from **wizardpanel-content** to **docked-to-container**. The Wizard theme is no longer necessary; the necessary design elements for the wizard are provided by the Axiom2018 skin.

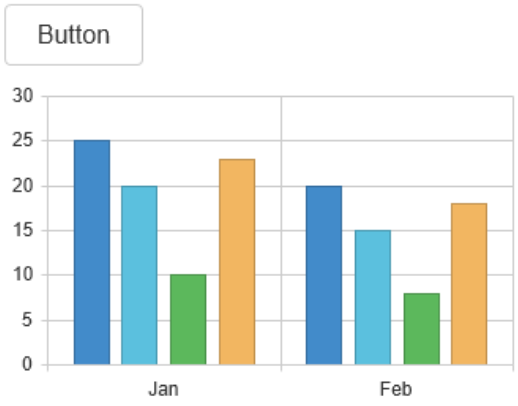
All other component-level styles from the legacy skin should be recognized by the Axiom2018 skin and continue to display as expected. However, it is a good idea to thoroughly review the form after changing the skin to detect any small formatting changes that may impact component display, and adjust the form accordingly.

The older the form, the greater the chance that additional manual adjustments will be required after changing the skin to Axiom2018. Older forms may use out-of-date designs that do not take advantage of the latest enhancements, and some of these out-of-date designs may impact the display. It is a good idea to review the entire form as part of the migration and identify any areas that could benefit from adopting new features.

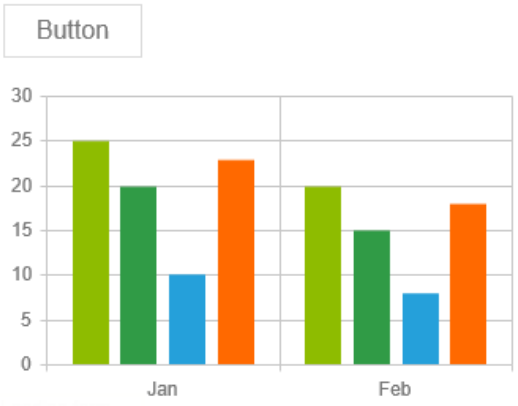
Lastly, if the form is old enough that it is still using a spreadsheet-formatted grid instead of a thematic grid, the grid should be updated as part of the migration. For more information, see Migrating spreadsheet-formatted grids to thematic grids.

Skin examples

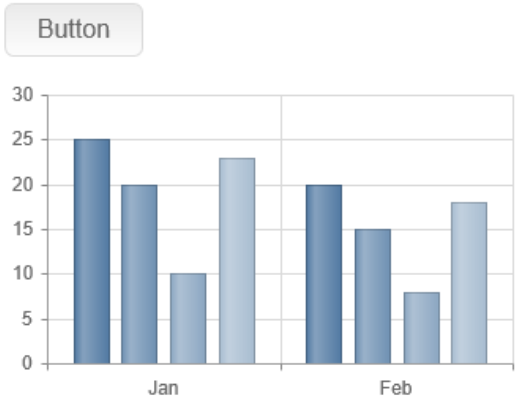
The following skin examples are provided to give an idea of the colors and styling used for each of the skins delivered with the Axiom Software platform.



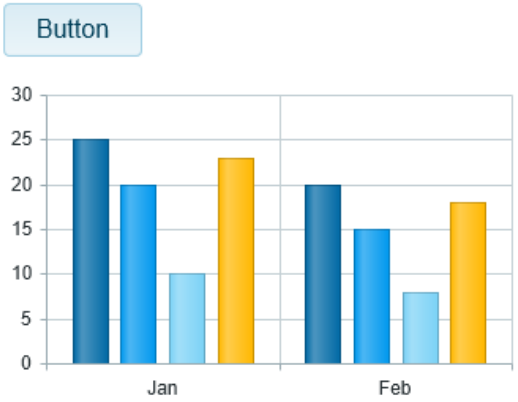
Axiom2018 and Axiom



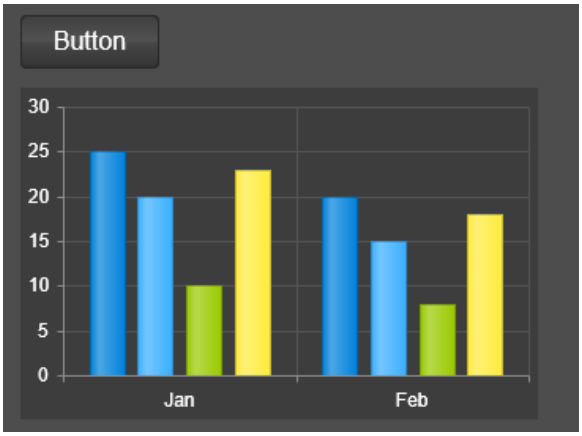
Metro



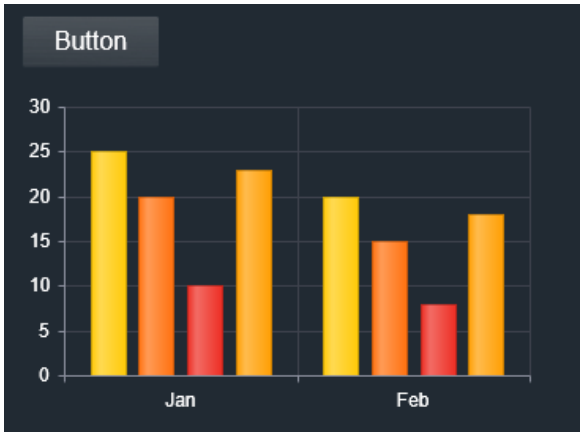
Uniform



Blue Opal



Black



Moonlight

Setting the theme for an Axiom form (deprecated)

IMPORTANT: Form themes have been deprecated in the default Axiom2018 skin. Themes only apply if your form uses a legacy skin (any skin other than the Axiom2018 skin). If a theme is specified at the form level (or at the component level) in a form that uses the Axiom2018 skin, that theme will be ignored.

When using legacy skins, the theme sets the overall formatting for the components in the form. It defines default formatting settings for components and also determines which styles are available for individual components.

The theme formatting is designed to fit particular use cases for forms. For example, the Report theme is intended for forms that display reporting data, whereas the Worksheet theme is intended for forms where users can edit data or make other user inputs.

When you first create a form, it does not have an assigned theme. If the theme is blank, the form uses default formatting settings, and default styles are available. The default settings may be sufficient for some forms. However, if the form falls into one of the theme categories then it is recommended to apply that theme to gain access to additional formats and styles, and to promote formatting consistency among certain form types.

Certain form designs will benefit more from themes than others. Some components have heavy interaction with themes and their dependent styles, while others do not. Forms using thematic Formatted Grid components should always have an assigned theme. Forms with heavy use of labels may also benefit. Other components, such as charts, do not currently have significant interaction with themes and styles, though this may continue to evolve in the future.

To set the theme for an Axiom form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, select the desired theme from the **Theme** drop-down list.

   For example, some of the available themes include:

Theme	Description
Home Page	Intended for use when designing form-enabled home pages. Formatting and styles are optimized toward this use case.
List	Intended for use when displaying lists of items in grids.
Report	Intended for forms that display reporting data. Formatting and styles are optimized toward displaying grids of data.
Grouped Report	Extends the Report theme to provide additional styles to display data using several levels of groupings.

Theme	Description
Worksheet	Intended for forms that collect data inputs. Formatting and styles are optimized toward displaying labels and input controls.
Dense Worksheet	Same as the Worksheet theme, but with less space between grid elements to accommodate forms that need to display a lot of information in a grid.
Sparse Worksheet	Same as the Worksheet theme, but with more space between grid elements to accommodate forms that display smaller amounts of information in a grid.
Wizard	Intended for forms that use the Wizard Panel component. Formatting and styles are optimized for the wizard environment.
Wizard Summary Report	Intended for forms that use the Wizard Panel component. Can be used to display a more compact summary of data than when using the regular Wizard theme.

Themes are evolving; you may see additional and/or altered themes.

TIP: You can also change the theme by editing the **Theme** field at the top of the Form Control Sheet. In this case you must type in the desired theme name.

NOTE: This list of themes represents the standard themes delivered with the Axiom Software platform. Packaged products installed for use with the Axiom Software platform may provide different themes, and/or these products may use modified platform themes. Additionally, it is possible to customize themes for your installation, although currently this process is not officially supported.

Themes and styles

The selected theme determines the styles available for use in the form, and the formatting of those styles. Fundamental styles are available for use in all or multiple themes, but the formatting used by the style may be different. For example, if you use the default style `row` in a Formatted Grid component, the row height for the style is different when using the Report theme versus the Worksheet theme.

Some themes have styles that are not available in other themes. For example, the Grouped Report theme provides additional styles that are not available in the Report theme. If you use one of these styles in the form and then switch the theme from Grouped Report to Report, that style will not be recognized by the new theme. In this case, an invalid style name is treated as no assigned style, which means the default formatting for the theme will be used.

Using component-level themes

By default, all components in the form use the theme set at the form level. However, in some cases, you may want certain components to use a different theme than the form-level theme. For example, the

form-level theme may be set to Worksheet, but you want a certain Formatted Grid component in the form to use the List theme.

You can set the theme at the component level by using the **Component Theme** property in the advanced component settings. To access this setting, click the **Show Advanced Settings** link underneath the Style property. Using the Component Theme property, you can assign any theme to the component. The Style property for the component will then reflect the styles for the component-level theme instead of the form-level theme.

Ideally, you should set the theme at the form level and then only use the Component Theme for components where you need to override that theme.

Using component styles

Styles are available at the component level to define certain formatting aspects of the component, such as fonts, borders, and positioning.

To use styles, you assign a named style to a component, such as **normal** or **page-title**. The component then uses the formatting settings defined for that style. For example, the page-title style could define formatting such as bold font, size 24, and blue color.

Each component can optionally be assigned one or more styles to define the formatting of that component. The available styles depend on the component type. If you are using a legacy skin instead of the default Axiom2018 skin, the available styles also depend on component's assigned theme (either inherited from the form-level theme or specified at the component level).

Additionally, many component types allow you to explicitly define certain formatting properties, such as font size or border width. You can define these properties in addition to using styles or instead of using styles. If the style and the component property are in conflict, the component property takes precedence.

IMPORTANT: This topic discusses how to apply component-level styles. If you want to apply row and column styles to a Grid data source for a Formatted Grid component, see Using row and column styles in a thematic grid.

Order of precedence for component formatting

For any particular formatting property, the settings used for a component are determined in the following order. If the top item in the list is not defined for a component, the next item in the list is used, and so on.

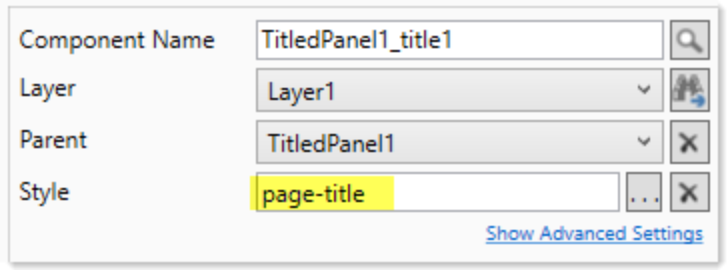
1. Specific formatting settings defined in the component properties.
2. Formatting defined by the style(s) assigned to the component. If multiple styles are assigned in a comma-separated list, the last-listed style takes precedence.
3. Formatting defined by the skin assigned to the form.

NOTE: If your form uses a legacy skin (any skin other than Axiom2018), the component formatting may also be affected by the theme specified at the form-level or the component level.

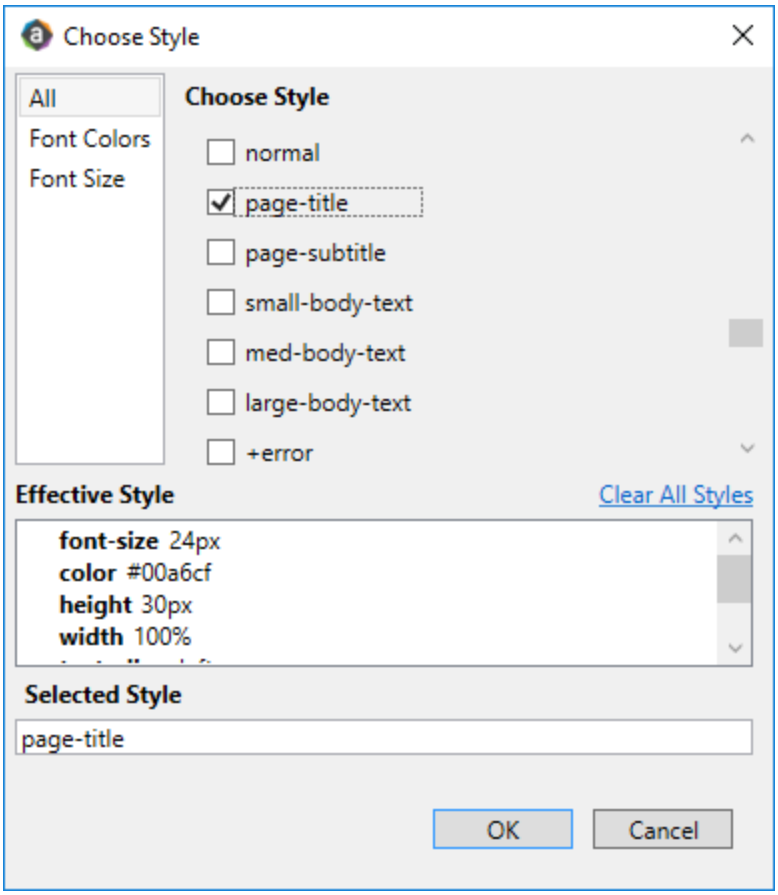
As an example, consider a Label component and the font size. If the font size is specified using the **Font Size** property in the advanced component settings, then that size is used. If not, then the font size specified by the assigned style for the component is used. If the component is not assigned a style, or if the assigned style does not specify a font size, then the font size specified by the skin is used.

Assigning styles to a component

To assign styles to a component, use the **Style** property in the component settings. If styles have already been assigned, they will display here.



Click the **Select component styles** button **[...]** to open the **Choose Style** dialog. From here, you can view the available styles for the component, assign styles, and view the effective formatting resulting from the selected styles.



- The **Choose Style** section lists the available styles for the component. Styles may be organized by categories, which you can select from the box to the left. For example, click **All** to view all styles, or click **Font Colors** to view only styles relating to font color. Additionally, some styles have descriptions that display in a tooltip when you hover your cursor over the style name.
- Select a base style (style without a plus sign) before selecting any add-on styles (styles with a plus sign). See the following section for more information about base styles and add-on styles.
- Once a style has been selected, the style name displays in the **Selected Style** box. If additional styles are selected, those styles will be added in a comma-separated list, with the last-selected style listed last. If the styles are in the wrong order for the desired order of precedence (see the previous section), you can clear the selections and then re-apply them in the desired order. When you click **OK**, the styles listed in the Selected Style box are written to the component properties.

Once one or more styles have been selected, the effective style for the component displays in the **Effective Style** box. This allows you to see precisely which formatting properties will be applied to the component based on your style selections. If a particular formatting property is not listed, then the selected styles do not affect that property. The formatting for any unlisted property will be determined by the skin, or by the individual component properties (if applicable and if defined for that component).

When styles are assigned to a component, you will see the effect of these styles in the Form Designer (and in the rendered form). If the component does not display as you expect (even after clicking **Refresh**), then check the following:

- If the component is assigned multiple styles, check the effective style properties to make sure the styles are applied in the correct order. You may need to change the order of styles to get the desired effect. Base styles should always be listed before add-on styles.
- Remember that any formatting specified in the component properties takes precedence over styles. If a component already has a defined formatting property such as font size, then any font size in the style will be ignored. In this case, you must clear the component-level property in order to use the formatting defined in the style.

  NOTE: If you apply a style to an existing component on the canvas, and the style contains size or position properties, then you must always use **Show Advanced Settings** to clear out the component's existing size and position properties before the style properties will take effect.

- Some components do not support certain formatting properties. If you apply a style that contains these unsupported properties to a component, then the style properties will either be ignored or not display as expected. For example, Combo Box components do not support borders (other than the built-in box border), so if you apply a border style to the component it will not display as expected. As a guideline, check the advanced properties of the component to see if the formatting property is available as a component-level property. If it is not available, then that property probably cannot be set by a style.

When a form is rendered, if a specified component style is not found then it is simply ignored. For example, this may occur if a style name was manually typed into the Form Control Sheet incorrectly.

Using base styles and add-on styles

Styles fall into two categories:

- *Base styles* define all of the necessary formatting to properly display the component, according to the purpose of the style.
- *Add-on styles* apply one specific formatting property (or a small group of related properties), such as to apply a font color. These styles are intended to layer on top of a base style, in order to modify that one specific property.

For example, a base style for Label components is **page-title** and an add-on style is **right**. If you have a label where you want to use the title formatting but you also want it to be right-aligned, then you would select **page-title,right** for the style. The second style of **right** only layers on the text alignment; it does not overwrite any of the other properties of the **page-title** style.

The base style should be listed first, followed by the add-on styles. In the Choose Style dialog, add-on styles are differentiated from base styles by a plus icon next to the style name.

Styles and Formatted Grid components

Formatted Grid components have a special application of styles:

- The **Style** property at the component level only affects the outer formatting of the component, such as the title bar and component border. It does not affect the grid contents.
- For thematic grids, additional styles can be applied at the row and column level, within the Grid data source. These styles affect the formatting of the grid contents. For more information on how to use row and column styles, see Using row and column styles in a thematic grid.
- For spreadsheet-formatted grids, the formatting of the grid contents is determined by the spreadsheet formatting. Styles do not apply to the grid contents.

Style FAQ

My selected style contains position / size properties, but my component isn't honoring them.

If position and size are set at the component level, these properties will override the style. When you drag a component on the canvas, position and size gets set automatically. So if you apply a style afterward, you must manually clear out the size and/or position properties on the component (use **Show Advanced Settings** to access them) in order to use the style properties.

I selected a style but it doesn't seem to do anything, or doesn't look right.

Some generic styles are available to all components, but these styles may contain settings that do not apply to some components or do not work well with some components. Also, some component-specific styles are intended for specific use cases. For example, the `titledpanel-body` style for panels works well in the Titled Panel use case, but otherwise may not be very useful when using panels in other form types.

Additionally, remember that style names are case-sensitive. If you have manually typed a style name instead of using the helper tools to choose styles, then you may have mistyped the name or used the wrong case. Use the style helper tools to verify the style name and case.

Why isn't there a style to do <something>?

Styles are continually evolving. Additional styles may be added in future releases. Remember that you can use the advanced component settings to set component-level formatting if a style is not available to do what you want to do.

General Design Concepts for Axiom Forms

This section discusses general design concepts for configuring Axiom forms, such as:

- Understanding how data is refreshed and saved for the form
- Linking form components to data sources
- Setting up interactivity within the form
- Saving to the database from the form
- Using button commands in the form

Update and save behavior for Axiom forms

This section explains the update and save-to-database behavior that occurs when a user views a file as an Axiom form. Axiom form designers should understand this behavior to set up their forms appropriately for user interaction.

NOTE: When using composite forms, special behavior applies to the form update cycle, as both the parent and child forms may be updated. For more information, see Form session and update behavior for composite forms.

Update behavior

The data displayed in an Axiom form is determined by the data queries set up in the source file. When a user views an Axiom form, the data in the form is updated as follows:

- When the user first opens the Axiom form, a calculation is performed and data is refreshed in two stages:
  - First, Axiom queries that are set to refresh on open are refreshed.
  - Then, the equivalent of a "manual" refresh occurs. This means all Axiom queries set to **Refresh on Manual Refresh** are refreshed.

  This determines the initial state of the Axiom form.

- If a user changes an interactive component on the Axiom form (such as a combo box or a check box), and the interactive component is configured to **Auto Submit**, then the current state of all interactive components is written back to the source file. A calculation is performed and all active Axiom queries that are configured to run on manual refresh are refreshed. The Axiom form is updated after this refresh is complete.

  NOTE: The changed state of the component is not saved within the source file, it is a temporary value for the current session. When the Axiom form is next opened, it starts with the default component values.

  If the interactive component is not configured to Auto Submit, then the user must refresh the Axiom form by using the Button component in order to submit changed values.

- If a user refreshes the Axiom form by using the Button component, then the current state of all interactive components is written back to the source file. A calculation is performed and all active Axiom queries that are configured to run on manual refresh are refreshed. The Axiom form is updated after this refresh is complete.

The user's security filters apply to the data queries, just as if the user had opened the source file directly and refreshed. All data displayed in the Axiom form will be specific to the current user (unless a component is displaying hard-coded data, or data left over from an inactive query).

If data lookups are used in the form-enabled file, they will be executed using the normal execution behavior for data lookups—such as executing on open, or executing after a particular Axiom query is run. Unnamed data lookups will be executed after Axiom queries whenever a form update occurs.

NOTE: If a user refreshes the web page for the Axiom form by using the browser refresh / reload functionality instead of using the Button component, this is interpreted as the user closing and reopening the file. The "file open" refresh behavior applies, and the Axiom form will revert to its initial state.

Save behavior

If the source file is configured to save to the database, then users can trigger a save-to-database from the Axiom form. The following actions can trigger a save-to-database:

- When a user changes an interactive component, and the component is configured to both **Auto Submit** and **Save on Submit**.
- When a user clicks the Button component, and the component is configured to **Save on Submit**.

In both cases, the update behavior occurs as described above—the current state of interactive components is sent to the source file, and the file is calculated and refreshed. After the refresh is complete, a save-to-database occurs. While the data is saved, a status message displays in the lower left-hand corner of the Axiom form.

When the save is complete, the source file is calculated again, and any Axiom queries that are set to **Refresh after save data** are run. The Axiom form is then updated, and a confirmation dialog informs the user that the save completed successfully.

NOTE: The source file itself is *not* saved when a save is triggered from an Axiom form, only a save-to-database occurs.

The user must have the **Allow Save Data** security permission for the file in order to perform the save-to-database. The user's security filters apply to the data save as normal.

Axiom form update process

The following diagram illustrates the full update process for Axiom forms. This is provided to help form designers understand the timing of various events, so that they can set up formulas and interrelationships between components and Axiom features set up in the file.

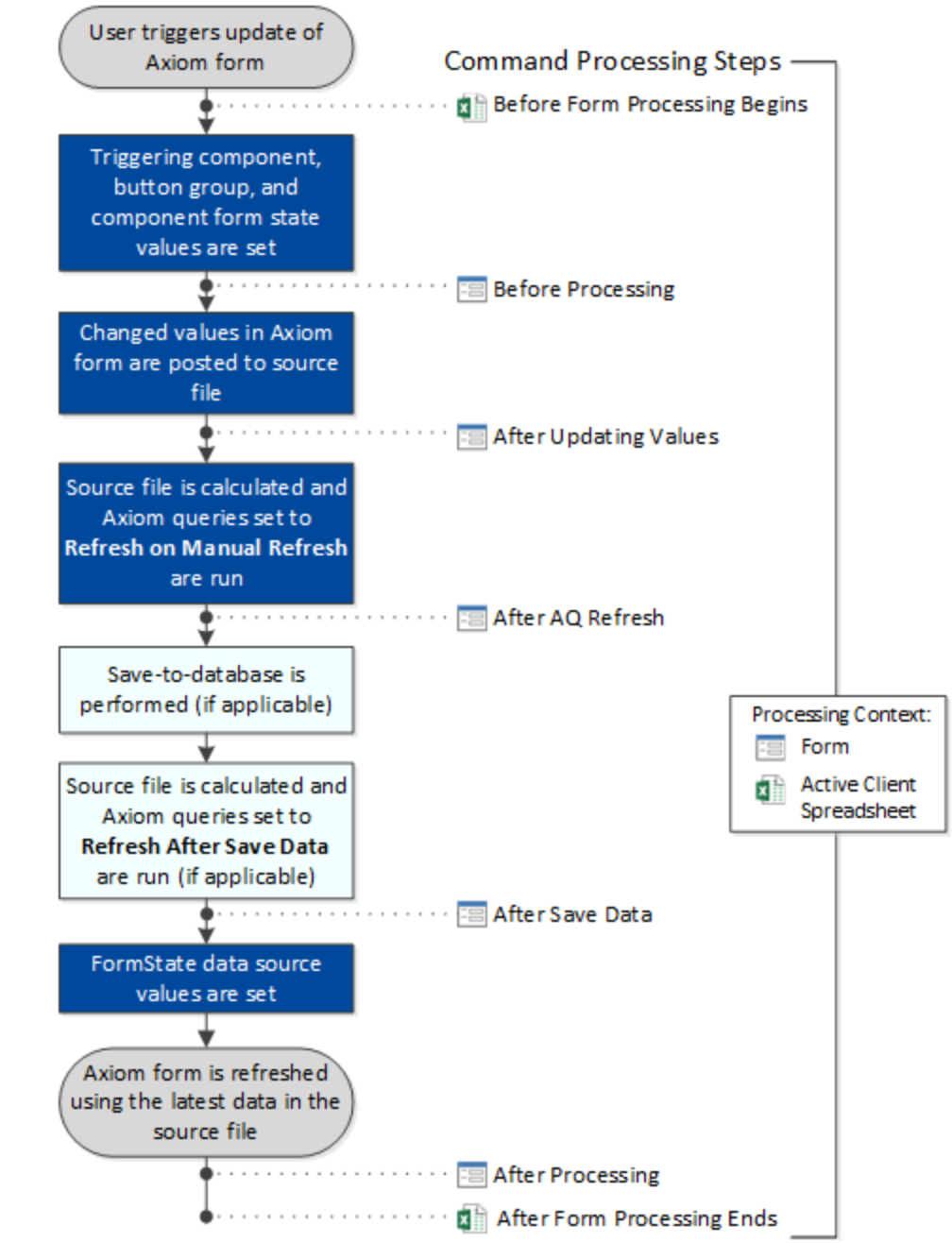
Users trigger this update process by doing one of the following:

- Clicking a Button component in the form.
- Changing any interactive component that is configured to Auto-Submit.

The Command Processing Steps only apply when the update is triggered by a Button component that has one or more commands. The processing steps indicate when the command will be executed during the process. When using an Axiom form as a dialog in the Excel Client or the Windows Client, you can also indicate where the command will be executed (the *processing context*)—either on the form itself, or on the active client spreadsheet.

NOTE: Refresh variables do not trigger a full form update as described in this topic. For more information on what actions occur when refresh variables are applied to a form, see Defining refresh variables for the Web Client Filters panel.

Axiom Form Update Process



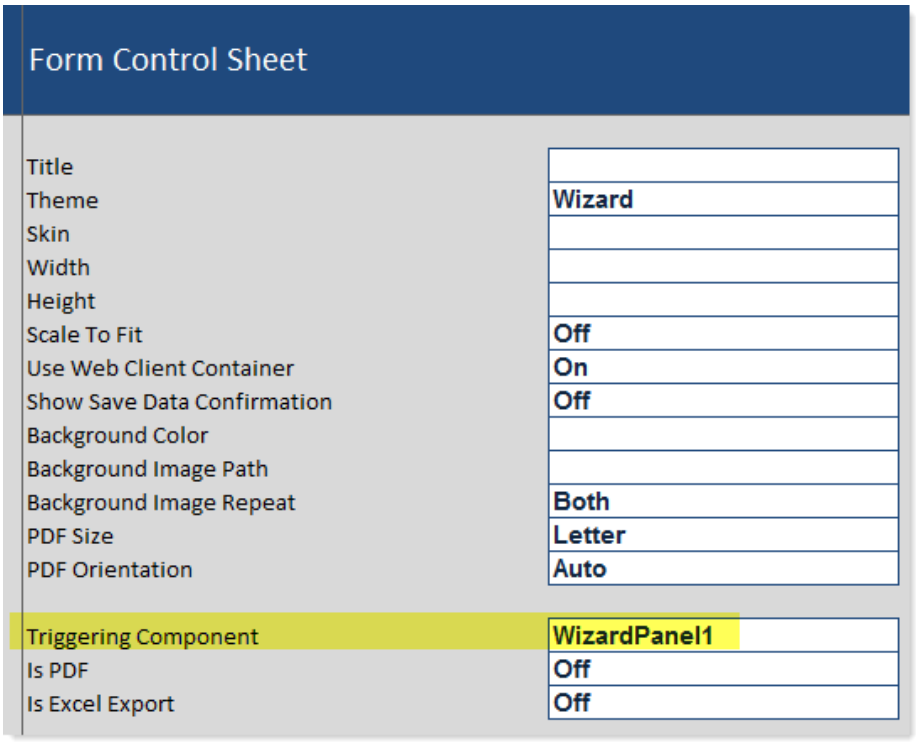
NOTES:

- The **After Save Data** processing step is only evaluated if a save is performed during the update. If you have a command that is configured to run After Save Data, but no save-to-database is performed, then the command will not be run.
- Data lookups will be run as normal (unnamed are run as part of every data refresh, named are run after specific Axiom queries if configured to do so).
- When using a composite forms, special behavior applies to the form update cycle, as both the parent and child forms may be updated. For more information, see Form session and update behavior for composite forms.
- Data Grid components are not refreshed by default during the form update cycle. Instead, the **Component Dependencies** property is used to create an association between the Data Grid component and one or more other components. If one of the other components submits a changed value, the Data Grid component is refreshed.

Keep in mind that refreshing the Axiom form using web browser functionality does not trigger this update process. Refreshing the browser page reloads the form as if it were initially opened. For more information on how an Axiom form determines its initial state when it is opened, see Update and save behavior for Axiom forms.

Referencing the triggering component of an Axiom form update

When an update is triggered for an Axiom form, Axiom Software writes the name of the specific component that triggered the update into the **Triggering Component** cell at the top of the Form Control Sheet.



For example, if the update is triggered by the user clicking a Button component, then the name of that button is written to the Triggering Component field. If the update is triggered by the user selecting an item from a Combo Box component, then the name of that combo box is written to the Triggering Component field. Each update of an Axiom form is always triggered by a single specific component—either a Button component, or an interactive component that is configured to Auto-Submit.

You can use the Triggering Component field to configure "targeted updates" of data in the Axiom form. For example, you may have an Axiom query that you only want to run when a particular component changes in the Axiom form. If any other component in the form changes, the Axiom query should not be run. You can do this by using a formula to dynamically enable or disable the Axiom query based on the value of the Triggering Component field.

The following example is a simple formula that could be used in the Active setting for the Axiom query. If the update is triggered by Button1, then the query is active and will be run. If the update is triggered by any other component in the form, then the query is inactive and will not run. The same kind of formula could also be used to enable or disable certain refresh behavior settings for the Axiom query, such as to turn **Refresh on Manual Refresh** on or off.

```
=If(Control_Form!D15="Button1","On","Off")
```

Special triggering component behaviors

In certain cases, the **Triggering Component** field is not populated with a user-defined component name. Instead, it populated with is a reserved name to indicate a special triggering component behavior. These reserved names are as follows:

- **Axiom.RefreshPanel**: This name indicates that a refresh was triggered by the Web Client filter panel. You can use this to dynamically enable or disable Axiom queries for the data refresh. A full form update does not occur in this situation, only a data refresh. For more information, see Defining refresh variables for the Web Client Filters panel.
- **$ParentForm**: This name is used in the child form to indicate that the form update was triggered by the parent form, instead of by a component in the child form. This only occurs in composite form configurations using the Embedded Form component. For more information, see Using composite forms.

  NOTE: There is no equivalent reserved name when the child form triggers a form update to the parent form. In that case, the triggering component for the parent form is logged as the Embedded Form component.

Setting up the source file for the Axiom form

This topic discusses design considerations for the source file for the Axiom form—the report file, template file, or other Axiom file that queries the data for the form and contains the form setup.

Querying data for the Axiom form

When you add components such as formatted grids or charts to an Axiom form, these components must be linked to data within the Axiom file. You can use any data query method to bring this data into your file, such as Axiom queries, data lookups, and Axiom functions. You can also "hard-code" data within a sheet as needed.

IMPORTANT: Whenever possible, Axiom queries and data lookups should be used instead of Axiom functions, for improved performance. GetData functions should be avoided unless there is no other way to return the required data for the form.

The data queries can be placed on any sheet in the file. Data is linked to Axiom form components by using data source tags, so it does not matter what the sheet name is, and whether or not it is set up on the default Control Sheet (although obviously you will need to do this if you want to use an Axiom query to obtain the data).

Any data that you want to display in the Axiom form must be queried or hard-coded in the file, and then flagged with the appropriate data source tags. For more information on tagging data in the file as data sources for Axiom form components, see Linking components to data.

Axiom query design considerations

When setting up Axiom queries within a form-enabled file, it is important to understand how these queries will be refreshed when the Axiom form is viewed. Queries should be configured so that they only run when it is necessary for them to run, to improve the performance of the form. For more information on this refresh behavior, see Update and save behavior for Axiom forms.

Refresh on open

When the Axiom form is opened, active Axiom queries are refreshed as follows:

- First, queries set to **Refresh on Open** are refreshed.
- Then, another refresh occurs that is equivalent to a manual refresh in an Axiom file (as if the user clicked the **Refresh** button). This means that all active Axiom queries that have **Refresh on Manual Refresh** enabled are run.

So if an active Axiom query has both of these refresh options enabled, it will be run twice when the file is initially opened. Generally speaking, refresh on open should only be enabled if refresh on manual refresh will be off when the file is initially opened (either hard-coded to off, or conditionally disabled using a formula).

Refresh on manual refresh

Any time the form is updated (either by an interactive component configured to **Auto-Submit** or by the user clicking a Button component), all active Axiom queries with **Refresh on Manual Refresh** enabled are run.

In most cases, it is not necessary to run a query this often, and doing so may significantly impact form performance. Ideally, if a query needs to be run after the form is opened, the refresh on manual refresh setting should be dynamically enabled and disabled using a formula. For example, the query might be configured to run or not depending on which component triggered the refresh (using the **Triggering Component** setting on the Form Control Sheet).

IMPORTANT: When troubleshooting an Axiom form, it is important to remember this refresh behavior. It may seem like something is not working, when in reality an Axiom query may be refreshing when it should not be, and overwriting changes that you have made to the source file.

Refresh after save

If you are saving data to the database from the form, the **Refresh after save** option can be used to run an Axiom query after this save occurs. For example, a user might input data into the form to be saved to the database, and then afterward you want this saved data to be available to form components such as a combo box or a formatted grid. Refreshing the query after the save-to-database will update these components with the relevant data.

Design for the Web Engine

When an end user views an Axiom form, a copy of the source file is opened by the Axiom Application Server using the Web spreadsheet engine. Therefore the source file cannot use any features that are not

supported by this engine. Use of any incompatible spreadsheet features will either be ignored or will result in an error.

Any files configured for use as Axiom forms should follow the same spreadsheet design considerations as the Windows Client. For more information, see the *Axiom File Setup Guide*.

Performance considerations

The design of the source file for the Axiom form should be as minimal as possible. The file should contain only those queries, formulas, and formatting that are necessary to drive the form data and functionality. If the file contains any content that is not necessary for the form, that content should be deleted.

When using remote server technology such as the Axiom Application Server, the Web spreadsheet engine is running on an application server. This engine consumes a certain amount of resources that would normally be used on the client machine. If you have 100 people looking at Axiom forms, your server will have 100 spreadsheets that it is processing on the server at the same time. The scalability of the server is far greater than the scalability of a client workstation, but the performance impact still should be considered. The faster your file is able to process, the more responsive the form experience will be.

Other design considerations

- If the source file is a file group file, and you are using a calc method library with the file, then the first two rows of the sheet associated with the calc method library should be reserved for system-generated validation codes (template and calc method validation).
- If you are designing form-enabled plan files, see also Designing composite plan files with Axiom forms.

Linking components to data

Certain components—such as formatted grids, combo boxes, and various charts—require data sources to define the data to be displayed in that component.

Data sources are defined in the source spreadsheet file and then linked to their associated component on the Axiom form canvas. This is accomplished by:

- Using reserved tags in the file to flag data as belonging to a particular data source.
- Configuring the component properties to use that data source.

This topic discusses the general process of how to tag data in a sheet and configure a component to use that data. You can use any means to bring the data into the sheet—such as Axiom queries, data lookups, GetData functions, or hard-coding data. For more information on designing the file to bring in data for an Axiom form, see Setting up the source file for the Axiom form.

Placing data source tags in a sheet

To define a data source for use in an Axiom form component, you must flag the data in the spreadsheet using reserved tags. The specific tags to use depend on the type of component. The following example shows the data source tags for a column chart. In this example, the data is hard-coded—the purpose of the example is simply to show the structure of the tags.

The general structure is as follows:

- A primary data source tag to define the data source, and to identify the control row and control column for the data source (B2 in this example).
- Row and column tags to identify the labels and the data to include in the data source, as well as other data source properties (column B and row 2 in this example).

	A	B	C	D	E	F	G	H	I	J
1										
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Column	Actuals	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Column	Budget	30000000	13000000	19000000	18000000	17400000	24000000

Example data source tags

IMPORTANT: All column and row tags must be placed underneath and to the right of the primary tag. Axiom Software will not recognize any row tags placed above the primary tag, or any column tags placed to the left of the primary tag. The column and row tags do not need to be contiguous; there can be blank columns and rows in between the tags. Axiom Software will stop reading column and row tags if it encounters a new primary tag or a tag that is not valid for the current primary tag (thus allowing data sources to be placed side by side or stacked on top of each other). The primary tag must be placed within the first 500 rows of the sheet.

You can manually type these tags, or you can use a wizard to automatically create tags for you. You can add the tags first and then populate the cells with data, or if the data already exists in the sheet then you can highlight the data to add the tags around it. To use the wizard:

- Right-click and select **Create Axiom Form Data Source**, and then select the type of data source. For example: **Create Axiom Form Data Source > Column Chart**.

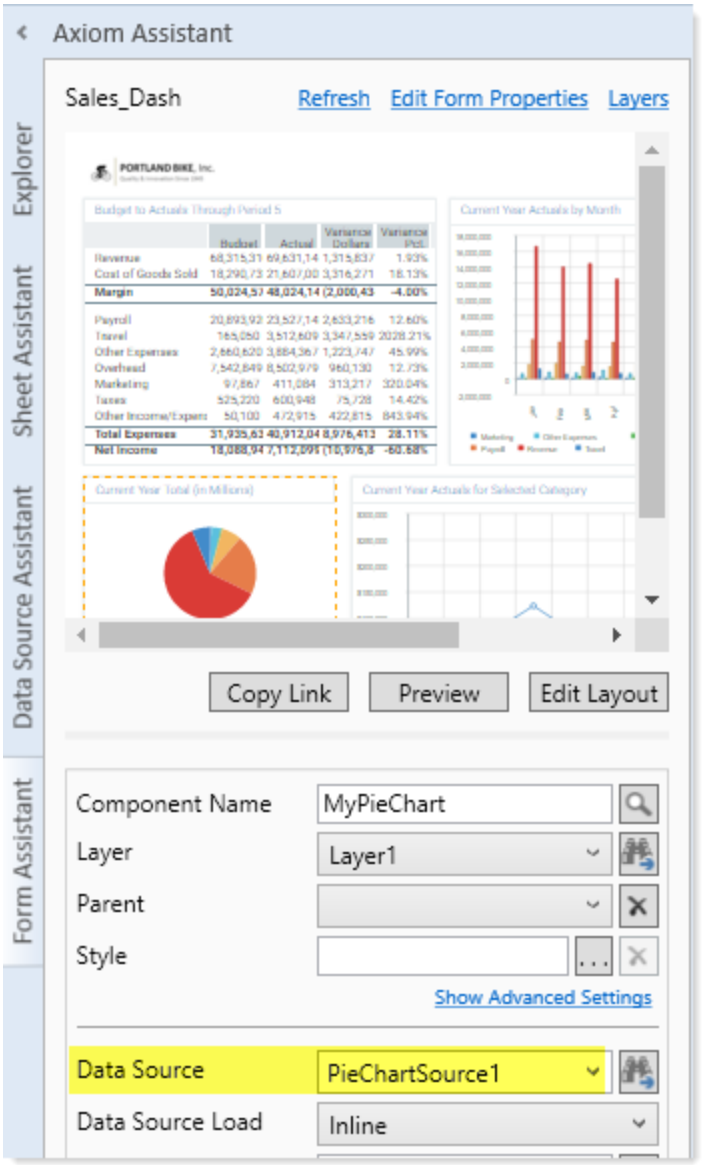
If you right-clicked on highlighted data, then the tags will be placed to the left and top of the selected data (these cells must be blank; the wizard will not overwrite any existing contents). The data source name in the primary tag will be generic, such as "ColumnChartSource1"—you can leave this or change it to something that better describes the data in the data source, such as "BudgetToActuals" in the example above.

If the data for the data source is populated using an Axiom query, and the query is set to rebuild or insert, then you should generate some of the tags using the in-sheet calc method, so that they will adjust dynamically to the data. For example, if the query is a standard vertical query, then the row tags should be generated using the in-sheet calc method. This means that as the query inserts rows into the data range, they will be tagged to be included in the data source.

For more information on the specific tag syntax for each component type, see the individual topic for each component. A list of all components with links to their individual topics can be found here: Axiom Form Components.

Specifying the data source for a component

Once the data source tags have been placed in the sheet, you can configure the component to use that data source. In the Form Assistant task pane, select the component in the canvas so that its properties display at the bottom of the pane. Then, set the **Data Source** property to the data source that you created.



Data sources display in the drop-down list using the name defined in the primary tag. Only data sources that are appropriate for the component type are listed (for example, only Pie Chart data sources display when configuring a Pie Chart).

NOTES:

- Bar Charts, Column Charts, Area Charts, Waterfall Charts, and Line Charts all use the same data source type—XYChart. All eligible XYChart data sources will display when configuring these charts, regardless of which component type you placed on the canvas.
- Scatter Charts, Scatter Line Charts, and Bubble Charts all use the same data source type—ScatterChart. All eligible ScatterChart data sources will display when configuring these charts, regardless of which component type you placed on the canvas.

You can also edit the component properties using the Form Designer.

If you are working with a component's properties and you want to find its corresponding data source in the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Using interactive components in an Axiom form

You can use interactive components in an Axiom form to collect inputs from users and then perform an action based on those inputs. For example, you can save the user's inputs and other data to the database, or you can change the contents of the form based on the user's inputs.

NOTE: If the only purpose of a user input is to change the data shown in the form, then you can use refresh variables as an alternative to setting up interactive components. See the following discussion Using interactive components versus refresh variables for more information.

Interactive components allow form users to change the state of the component in some way. The user may be able to select a value from a list, or type text into a text box, or simply toggle the component as selected or not selected. After the user interacts with the component to change its state, this state change is submitted back to the form source file. The form can then trigger an action and/or change in some way based on the user's interaction with the component.

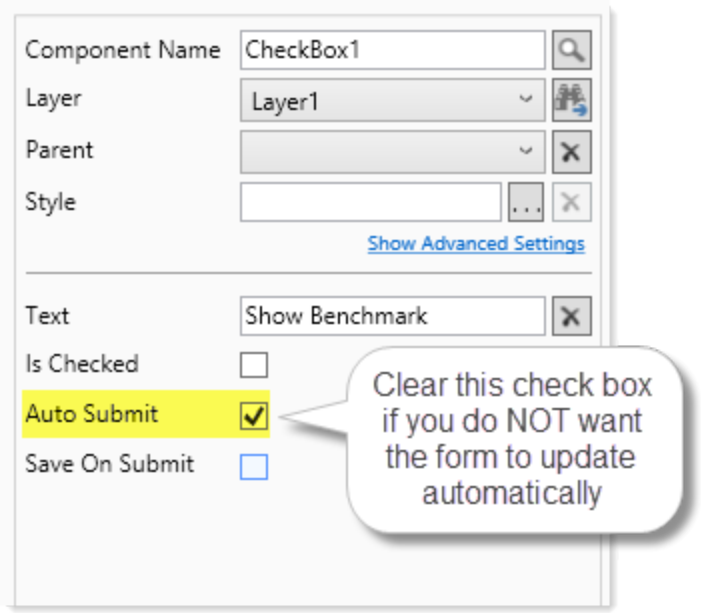
How interactive components work

All interactive components have a dedicated property that reflects the current state of the component. For example, the CheckBox component has the property **Is Checked** on the Form Control Sheet. When a user first opens an Axiom form with a CheckBox component, the check box displays using the state of the Is Checked property as saved in the source file. For example, if the source file for the Axiom form has Is Checked set to Off, then the check box is not checked when the user opens the form.

The user can then interact with the check box to change it from unchecked to checked. This state change is submitted back to the source file, and the Is Checked property is changed from Off to On as a result of the user's interaction. The source file is then refreshed, and the form is updated.

In order for the content of the Axiom form to change based on this user interaction, a second component must reference the state of the interactive component and then change in some way based on that state. For example, a series tag for a column chart could reference the check box state to determine if the series is visible or not. The key point is that the Axiom form interactivity is completely dependent on the file setup. If no other components in the Axiom form reference the state of the interactive component, then no change will occur in the form (other than the check box being checked). It is up to the Axiom form designer to set up the interactivity as desired.

By default, most interactive components are configured to automatically update the Axiom form when their state changes. For example, when a user selects or clears a check box, the changed state is automatically written back to the source file, a refresh occurs, and then the form is updated. If desired, you can change this configuration so that the Axiom form will not update until the user manually initiates an update by using a Button component. You may want to do this if your Axiom form has multiple interactive components—for example, if you have multiple combo boxes, and you want all of the user's selections to be sent back to the source file at one time, rather than as each one is selected. To disable the automatic update for an interactive component, clear the **Auto Submit** setting for the component.



When a user interacts with a component to change its state, the state change that is submitted back to the source file is never persisted past the current session. This is because Axiom forms are always open as read-only. Although you can trigger a save-to-database from an Axiom form, you cannot save the file itself. The next time the form is opened, the component state will revert to its original state as saved in the source file. If you do need to persist the state change of a component, then you would have to save the component state to the database, and then query that saved state back into the source file (using a function or a refresh on open query) to set the initial state of the component.

Submit behavior

When a user opens an Axiom form, a copy of the source file for that form is opened on the Axiom Application Server. The form is rendered based on the settings in that source file. As the user interacts with the form, their changes are submitted back to the source file on the application server. The source file is calculated and refreshed, and then the user's form is then updated based on the current state of the source file. For more information, see How Axiom forms are rendered to users and Update and save behavior for Axiom forms.

It is important to understand that whenever an update is triggered, the current states of *all* interactive components may be written back to the source file, not just the state of the component configured to Auto Submit. There is no way to ensure that just one particular component will be submitted. In most cases only changed components will be submitted for performance reasons, but certain conditions may require the form to send the state of all interactive components, to ensure that the file copy on the application server remains in sync with the rendered form that the user is interacting with.

This behavior means that if you use a formula in an interactive property, you can only count on using the formula to set the initial state of the component. If the user interacts with the component, or if the form detects that it needs to submit the current state of all interactive components, then that formula will be overwritten with the current state of the component.

For example, if you use a formula to set the **Is Checked** property of a check box, the check box will start off using the result of that formula. Once the user interacts with the check box, the current state is submitted to the source file and the formula is overwritten with either True or False. Additionally, even if a user never interacts with the check box, the current state of the check box may be submitted anyway when other interactive components are changed and submitted, to ensure that the form and source file remain in sync.

There are ways to work around this behavior if you need to use a formula to change the state of a component during an already active session. For example, you can use indirect cell references with most components so that the interactive value is read from and written to a cell in another sheet rather than the property cell on the Form Control Sheet. To continue the check box example, you would enter something like `[Values!F5]` in the Is Checked property, to "redirect" the interactive value to that cell. You could then use action codes on the Values sheet to copy the result of a formula to that designated cell.

NOTE: When you are working on a source file in the Desktop Client and you preview the form, remember that the form is using a copy of the source file on the application server, not the copy that you have open in the Desktop Client. If you change interactive components in the preview, you will not see those changes reflected in the file that you have open in the Desktop Client.

Interactive components

The following components can be used as interactive components to change the Axiom form in some way:

- **Area / Bar / Column / Line / Waterfall Charts**: The selected item in these charts can be sent back to the source file.
- **Button**: Button components support several types of interactivity: 1) The currently selected button in a button group can be sent back to the source file (like a radio button), 2) The button can be used to launch a multi-select dialog and send the selections back to the source file, and 3) The button can be used to trigger an update of the form (including running specified commands).
- **Check Box**: The state of the check box (checked or unchecked) can be sent back to the source file.
- **Combo Box**: The selected item in the combo box can be sent back to the source file.
- **Data Grid**: The selected row in the grid can be sent back to the source file. Additionally, icons can optionally be used in the grid to execute commands, which can trigger an update of the form.
- **Date Picker**: The selected date can be sent back to the source file.
- **Formatted Grid**: Formatted grids support several types of interactivity: 1) The selected row in the grid can be sent back to the source file, 2) The changed value in an editable cell can be sent back to the source file, and 3) Various content tags can be used to display interactive controls in the grid and send information back to the source file.
- **Hierarchy Chart**: The selected node in the hierarchy can be sent back to the source file.
- **KPI Panel**: The selected KPI in the panel can be sent back to the source file. Additionally, each KPI can optionally be associated with one or more commands, which can trigger an update of the form.
- **Map View**: The selected pin, circle, or feature can be sent back to the source file.
- **Menu**: The selected ID in the menu can be sent back to the source file.
- **Pie Chart**: The selected slice in the pie chart can be sent back to the source file.
- **Radio Button**: The currently selected button in a button group can be sent back to the source file.
- **Slider**: The selected value on the slider can be sent back to the source file.
- **Text Box**: The text entered into the text box can be sent back to the source file.
- **Wizard Panel**: The value for the current step can be sent back to the source file, to drive the content for the current step.

Most interactive components store their current state in a designated property on the Form Control Sheet. However, some interactive features use different approaches, such as the target cell for interactive controls in a formatted grid. For more information on each component and how its current state is written back to the form source file, see Axiom Form Components.

Using interactive components versus refresh variables

Refresh variables can also be used in form-enabled files to provide interactivity. Form users can use the Filters panel to make selections for those variables and send the values back to the source file. The queries in the file are then refreshed. Assuming the queries reference the variable values in some way,

the data shown in the form can then change based on the user's selections for the refresh variables. For more information, see Defining refresh variables for the Web Client Filters panel.

Refresh variables are intended to impact the data shown in a form. They are best suited to provide interactivity for web-enabled reports, to impact the data refresh. Refresh variables are not suited for other interactive uses, such as to collect data inputs from the user to save to the database.

When deciding whether to use refresh variables or interactive components to affect the data refresh of a form, keep in mind the following differences:

- Interactive components are displayed directly on the form, along with any data shown. Refresh variables are displayed "on demand" when the user chooses to open and use the Filters panel.
- Interactive components always trigger a full update cycle for the Axiom form when their values are changed. Refresh variables do not trigger a full update cycle. The only updates performed are to submit the refresh variable values back to the source file and then refresh the data in the source file.

Saving data from an Axiom form

You can save data back to the Axiom Software database from an Axiom form. For example, you might want to allow users to save comments about the data in the form, or use the form as a data input tool. Only data saves are supported when using Axiom forms; the source file itself is not saved.

Configuring an Axiom form to save data

To set up an Axiom form to save data, you must do the following:

- Configure the source file for the Axiom form to save to the database. This is the same setup that you would use for any Axiom file—for example, to enable Save Type 1 for a report or for a plan file template. However, note that you may not want to enable Zero on Save in this environment, since data from previous saves are not persistent in the file.
- Configure the Axiom form with interactive components to collect user input, such as a combo box or a text box, or a formatted grid with one or more editable cells. The save-to-database process and the interactive component must interact in some way in the source file, so that when the user changes the interactive component in the Axiom form and the updated state is submitted back to the source file, the data to be saved to the database updates in response.

- Configure the Axiom form to trigger a save-to-database when updated data is submitted to the source file. You can do this in two different ways, depending on how you want the data to be collected and saved.
    - If the save should occur after a particular component is changed, then that component should be configured to **Auto-Submit** and to **Save on Submit**. For example, if you want the save to occur after the user has input text into a particular text box.
    - If the save should only occur when the user decides that they are ready to save, then you should use a Button component that is configured to **Save on Submit**. For example, if you are using a formatted grid where the user is entering inputs into multiple cells, you probably don't want to trigger a save after each input—instead you want to wait until the user has completed all of their inputs, and then the user can click the button to save their changes.

  When Save on Submit is enabled, the save-to-database always occurs *after* values are submitted back to the source file and a full refresh is performed. For more information on the order of the process, see Update and save behavior for Axiom forms.

IMPORTANT: The source file for the Axiom form is *not* saved when a save-to-database is triggered from the Axiom form—only data can be saved when using this environment. Therefore if the data saved by the user is expected to be displayed in the Axiom form, then you must configure the form so that any saved data will be queried back into the sheet via Axiom queries or GetData functions.

If an Axiom query in the file would be impacted by the save-to-database, you can use the refresh option **Refresh after save data** to automatically refresh the query after the save occurs. For example, if the save-to-database adds or updates a record in a particular table, and you are using an Axiom query to display the contents of that table in the Axiom form, you would want to enable this option so that the user does not have to manually refresh the form to see the changed data. Axiom functions such as GetData are automatically calculated after a save-to-database and do not need any special settings to display the changed data.

NOTES:

- The user viewing the Axiom form must have **Allow Save Data** rights for the source file in order to save data from the form. If the source file for the Axiom form is a plan file, process management (and workflow) ownership rights are honored to determine whether the user can save data.
- When a user views a file as an Axiom form, the source file is opened read-only (but still permits saving data) and no lock is placed on the file. Therefore it is possible for two users to view the same Axiom form concurrently and save data, and the last user to save in this scenario would overwrite the first user's changes (if both users are saving to the same keys).
- When saving data from a composite forms, additional considerations apply. For more information, see Saving data from composite forms.

Save-to-database example

The Axiom form contains a text box where a user can type in a comment about the data in the form. To enable this to save to the database, it could be set up as follows:

- Set up Save Type 1 in the source file to save the comment to the appropriate place in the database. You may be able to assume the necessary keys for the save within the file, or the user may need to specify the appropriate keys in the Axiom form—for example, using other text boxes or, more likely, combo boxes that present the appropriate choices of department and account (or whatever the keys are in this scenario).
- Set up a Text Box component within the Axiom form, where the user will enter their comment. You may also want to define some initial text for the Text property; in this case, something like "Enter a comment..."
- Set up a Button component within the Axiom form, and enable **Save on Submit** for the button. You may want to change the text of the button to something like "Save" or "Update." (Optionally you could configure the text box to **Auto-Submit** and **Save on Submit** instead, but only if it is a single-line text box.)
- In the source file, in the cell that is configured to save the comment to the database, enter a cell reference to the Text property of the component. Since you probably don't want to save the text "Enter a comment..." to the database, you could set up the save tag for that row to use an IF statement.

  Alternatively, you can use an indirect cell reference in the Text property of the component, so that it points to the desired cell—such as [Sheet1!C15]. When using an indirect cell reference, the Text Box component displays the text from the designated cell, and writes the updated text to the designated cell (instead of reading and writing to the Text property field directly).

When the user views the Axiom form, they will see the text box with the text "Enter a comment...". They can choose to type a comment into the text box. When they click the button to submit, then the comment will be written back to the Text cell of the text box settings (or to the designated cell in the file if an indirect cell reference is used). The file is refreshed and then a save-to-database is performed. The comment is now saved to the database.

It is important to keep in mind that the source file is *not* saved; only the data is saved. The next time this user opened the same Axiom form, they would see the default "Enter a comment..." again. To work around this, you could configure the Text property of the text box with a GetData function that queries the appropriate value from the database, so that the next time the user logs in, they would see the comment that they previously saved to the database. The GetData function could be wrapped in a conditional formula so that if the comment in the database was currently blank, the text "Enter a comment..." would display instead.

Component enablement and saving data

If **Save on Submit** is enabled for a component, then that component will only be enabled in the form if a save-to-database process is enabled in the source file on the Control Sheet (and if the user has **Allow**

Save Data security permissions to the file). You can dynamically enable or disable the save-to-database based on some other condition (for example, all required fields have inputs) to enable the component and therefore allow the user to save.

If a save-to-database process is enabled in the source file, and if changed values have been submitted to the form but no save has yet occurred, then a warning message will display to the user if they attempt to close the form. This warning message informs the user that they have made changes in the form that will not be saved. The user can choose to return to the form to complete their save, or continue closing the form.

NOTE: This "unsaved changes" warning will occur even if no components currently have **Save on Submit** enabled, or if a component does have **Save on Submit** enabled but the component itself is not currently enabled. The only way to avoid the warning on form close is to disable the save-to-database process up until the point where the user has completed enough inputs where the save would be valid and the user should be warned about losing their data.

User messaging when saving data

There are several layers of user messaging available when saving data to the database from an Axiom form. Some of these layers can be modified at the Axiom form level.

Messaging	Description
Prompt the user before saving	If you want to prompt the user to confirm that they want to perform the save, then you can define a **Confirmation Message** for the Button component. When the user clicks the button, the defined message will display and the user will have the option to continue or cancel. If canceled, then the form is not refreshed and no save-to-database occurs. This option is only available when using a Button component to perform the save. If you are using an interactive component with **Auto-Submit** and **Save on Submit**, then there is no opportunity to prompt the user before saving; the refresh and save will occur automatically.

Messaging	Description
Display confirmation of successful save	By default, if the save-to-database is successful, no confirmation dialog displays to the user. The user's only indication of the save-to-database is the yellow status message that displays in the lower left-hand corner of the form while the save is processing. If desired, you can configure the form so that an explicit confirmation message does display after a successful save. In the Form Assistant task pane or the Form Designer dialog, click **Edit Form Properties** to open the Form Properties dialog, and then select the check box for **Save Data Confirmation**. (Alternatively, this setting can be enabled or disabled on the Form Control Sheet using the **Show Save Data Confirmation** property.) If this setting is enabled, then a save-to-database confirmation message displays to the user in a dialog. This message simply informs the user that the save completed successfully; it does not contain any details about records saved. The user must dismiss the dialog to return to the form.
Display error messages from the save	If any errors occur during the save process, these error messages display to the user in a dialog. The error messages are similar to those displayed in the Excel Client and the Windows Client, however, the specific cell addresses are not displayed because they would not be meaningful to the Axiom form user.
Display warning message for unsaved changes	If changes have been made in the Axiom form but no save has occurred, a warning message displays to the user if they attempt to close the Axiom form. The user can choose to continue closing the form or return to the form. This warning only occurs if both of the following are true: • The form has a component that is configured to **Save on Submit**. • Changed values have been submitted back to the source file but no save has been performed (in other words, the source file has unsaved changes). If changes have been made in the Axiom form but none of those changes have been submitted back to the source file, then no warning message will display because Axiom Software is not aware of any changes.

Controlling component visibility and enabled status

When designing an Axiom form, you may want to dynamically control whether a particular component is visible and/or whether that component is enabled. You can do this by using a formula in the following properties for the component:

- **Visible**: Controls whether the component is visible.
- **Enabled**: Controls whether the component is enabled. Only supported for certain components.

By default, both of these settings are set to On, meaning the component is visible and enabled. If you want to modify either of these settings, you must do so on the Form Control Sheet (because this is the

only way to enter formulas into component settings). These settings do not display in the Form Assistant or the Form Designer.

To easily find these settings for a particular component, you can use the Form Assistant task pane. Select the component in the thumbnail, and then click the Show Control Sheet link to jump directly to settings for that component in the Form Control Sheet. The Visible and Enabled settings are located at the top of the component's settings, after the Component Name.

Showing or hiding a component

You may want to dynamically hide or show a component based on other selections made in a form. For example, you could have a check box that shows or hides a chart. In the **Visible** setting for the chart, you would enter a formula something like:

```
=IF(Control_Form!D45="On","On","Off")
```

Where D45 is the location of the **Is Checked** setting for the check box on the Form Control Sheet. If this is set to On (meaning the check box is checked), then the Visible setting for the chart will be set to On and the chart will be visible. If the check box is set to Off (unchecked), then the Visible setting for the chart will be set to Off and the chart will be hidden.

If you have several components that you want to make visible or hidden as a group, then you can place all of these components on a dedicated layer, or make them all child components of a Panel component. You can then use the Visible setting on the layer or the panel to show or hide all of those components at once. This approach is often used to switch between different "pages" or "screens" within a form—by grouping related components on a layer or a panel, and then showing or hiding the layer or panel as appropriate.

In some cases you may want to show or hide a component depending on where the form is being viewed. You can use the `GetDocumentInfo("EmbeddedFormType")` function to return the current context of the form. This function has the following return values:

- <Blank>: The form is open in a browser (Web Client). The function will also return blank while you are working within the source file for the form, or if the function is used in any non-form-enabled files.
- `Tab`: The form is open as a web tab in the Excel Client or Windows Client.
- `Dialog`: The form is open as a modal dialog in the Excel Client or Windows Client.
- `TaskPane`: The form is open as a task pane in the Excel Client or Windows Client.

Enabling or disabling a component

You may want to dynamically enable or disable a component based on other selections made in a form. If a component is disabled, then it displays as grayed out and users cannot interact with it.

For example, the form may have a button that the user can click to update a grid of data in the form, but you only want this button to be available after the user has made a selection from a particular combo box. In the **Enabled** setting for the button, you would enter a formula something like:

```
=IF(Control_Form!D53="","Off","On")
```

Where D53 is the location of the **Selected Value** setting for the combo box on the Form Control Sheet. If this is blank (meaning no selection has been made yet), then the Enabled setting for the button will be set to Off and the button will be disabled. If the combo box has a selected value (is not blank), then the Enabled setting for the button will be set to On and the user can click the button.

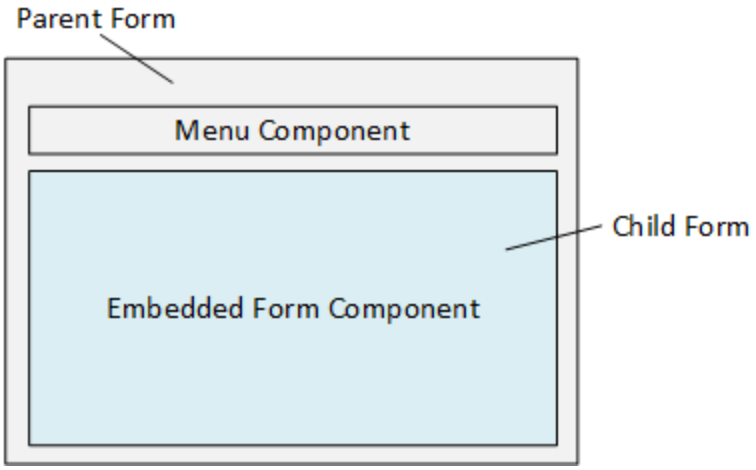
The following components support the Enabled setting: Button, Check Box, Combo Box, Hyperlink, Radio Button, Text Box.

Using composite forms

A *composite form* is a form that is built using multiple files instead of a single file. Composite forms consist of the following:

- A parent form that contains an Embedded Form component. The parent form can contain other components and otherwise works as normal, except that the Embedded Form component is used to open other child forms within the parent form instance.
- One or more child forms that display within the Embedded Form component. The Embedded Form component can be set up to display a single target child form, or it can be set up to dynamically switch between multiple child forms by using the Menu component. As the user selects items from the menu, the Embedded Form component displays the corresponding child form for that menu item.

When a user views the parent form, the child form displays as "embedded" within the parent, so that both the parent and child forms are displayed within the same browser page and share the same form instance.



Typical composite form setup

This feature makes it possible to take several individual form files and present them as a single unified "page" to the form user, instead of requiring the form user to navigate between the separate forms. This

approach can also simplify form design, by allowing different "screens" or features within a form to be created and maintained within separate files, instead of requiring all aspects of a form to reside within a single file.

NOTE: The primary composite form design assumes a single Embedded Form component and a Menu component to switch between multiple child forms. Although it is possible to use multiple Embedded Form components to display multiple child forms concurrently, there are some limitations. For more information, see Embedded Form component.

For example, you may want to create a web-based utility that allows users to perform several different but related actions. Each available action requires different screens, with components and data specific to that action. Instead of building all of the functionality within a single file, you can create different child files for each action, and then create a parent file that provides the overall user interface for selecting the desired action and displaying the relevant child form. Instead of one large, complicated file with many form layers and many sheets of data, you can have several lightweight files that are easier to build, modify, and troubleshoot. The child files can also be reused separately or in different contexts as appropriate.

Other potential use cases for composite forms include:

- Form-enabled plan files with multiple screens of information and inputs. The plan file template can provide the overall titles and menu for selecting screens, and then each screen can be sourced from individual utility files in the file group. For more information on this type of plan file design, see Designing composite plan files with Axiom forms.
- A web-based "driver manager" utility, where the parent file provides a menu for selecting driver files, and then each individual driver file displays as a child form within the parent. Instead of needing to open and review each driver file individually, the user can easily access and edit all drivers as needed.
- Dashboards with several different screens of charts and other visualizations. Each dashboard screen can be sourced from a separate child form, while the parent form provides the means to switch between these screens. If desired, the individual child files could also be accessed directly for different sets of users who only need to see that particular area or slice of the data.

A composite form uses a *shared form instance*, where the parent form and its embedded child forms are managed together by the Axiom Application Server as a related set of forms. If the Embedded Form component is used to display multiple forms (via a Menu component), all of those forms become part of the shared form instance. This shared form instance provides the following benefits:

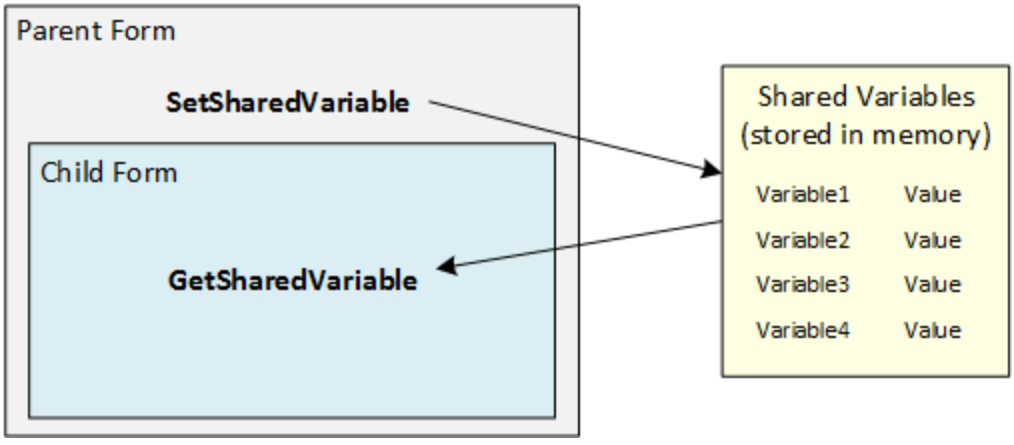
- **Shared variables:** All forms in the shared instance have access to a shared set of variables. You can set a value in one form and then reference that value in all of the other forms within the instance.
- **Persisted child forms:** Users can switch between viewing different embedded child forms, and the current state of each child form will be persisted within the shared form instance. All forms in the shared form instance are maintained on the server for the current session, so that as long as the parent form remains open, users can return to the child forms at any time and the state of each child form will be remembered.
- **Shared data cache:** All forms in the shared instance use the same GetData cache. If a GetData query is duplicated across multiple forms, after the first query the result can be retrieved from the shared cache instead of initiating another query to the database.

Sharing variables between parent and child forms

You can share a set of variable values between the parent and child forms that make up a composite form. These variables are known as *shared variables*, and they only apply when using the Embedded Form component to create a composite form. This feature allows all forms within the shared form instance to remain in sync for certain important values.

When a user views a composite form, Axiom Software maintains a single set of shared variables for all forms within the shared form instance. When any form in the instance needs to look up the current value for a specified variable, it looks up the value from this shared list. This shared list is stored in memory by the Axiom Application Server.

Shared variable values can flow in either direction. The variable value can be set in the parent form and then referenced by the child forms, or it can be set in a child form and then referenced by the parent (and other child forms).



Example shared variable flow

Variables and their values can be defined using the following methods:

- **SetSharedVariable function:** Each time this function is evaluated, it sets the current value of a named shared variable to the value defined in the function, overwriting any existing value for the variable. This method takes precedence over any other method of setting the variable value.
- **Interactive component:** An interactive form component, such as a combo box, can be configured to store its value using a named shared variable. Each time the component value is submitted to the source file, it sets the current value of the shared variable to the component value, overwriting any existing value for the variable.
- **Apply Shared Variable command**: A button in an Axiom form can be configured to use this command, which sets the value for one or more named shared variables. When a user clicks the button, the variables are set to the specified values, overwriting any existing value for the variables.
- **GetSharedVariable function:** If no value currently exists for a named shared variable when this function is evaluated, then the value of the variable is set to the value defined in the DefaultValue parameter of the function. This approach only works once, when the variable has no defined value. Once the variable has a value, the DefaultValue parameter is no longer evaluated and cannot affect the value of the variable.

Regardless of how the variable value is defined, the GetSharedVariable function can be used to reference the value in any form within the shared form instance.

It is recommended to use a dedicated sheet in each file to define and/or reference the shared variables. For example, the parent and child files could all contain a sheet named SharedVariables. In the parent file this sheet might contain SetSharedVariable functions to set the values, and then in the child files the sheet would contain GetSharedVariable functions to reference the values. Or the sheets might contain a mix of functions if some variable values are set in the parent file and other variable values are set in the child files.

This dedicated sheet is not required, but it simplifies form design to have all shared variables listed in a single known place within all of the related files. If instead the functions are scattered around different sheets, then it may become difficult to locate, review, and troubleshoot the variables.

NOTE: If any form in a shared form instance changes the value of a shared variable, the value is immediately available to the other forms. However, a form update must be triggered in the other forms in order to read the new value and refresh the form display. If a form is not updated after the value has changed, it will continue to reflect the previous value for the variable. Keep in mind that triggering an update in any child or parent form does not automatically cause the other forms to be updated. For more information on the update behavior for the forms in a shared form instance, see Form session and update behavior for composite forms.

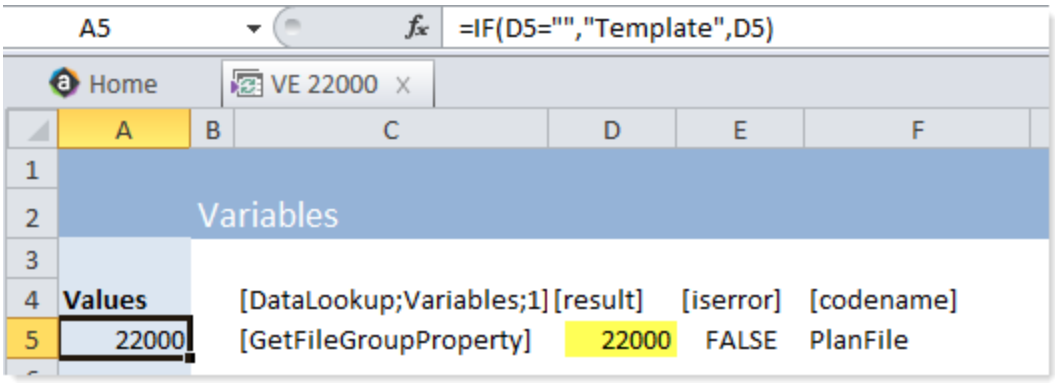
Defining shared variables using SetSharedVariable

The SetSharedVariable function sets the value of a named shared variable whenever the function is calculated. You should use the SetSharedVariable function when the value for the variable is derived from a data query, or by using calculations within the form source file. The SetSharedVariable function uses the following syntax:

```
SetSharedVariable("VariableName","VariableValue")
```

For example, imagine that the parent form is a template that will be used to create form-enabled plan files. The plan file contents are sourced from a series of child forms (utility files) displayed within an Embedded Form component. When a child utility file is opened in the Embedded Form component, it needs to know the current plan code for the parent plan file, so that the child form can retrieve the appropriate data for the plan code and save back data to the appropriate plan code.

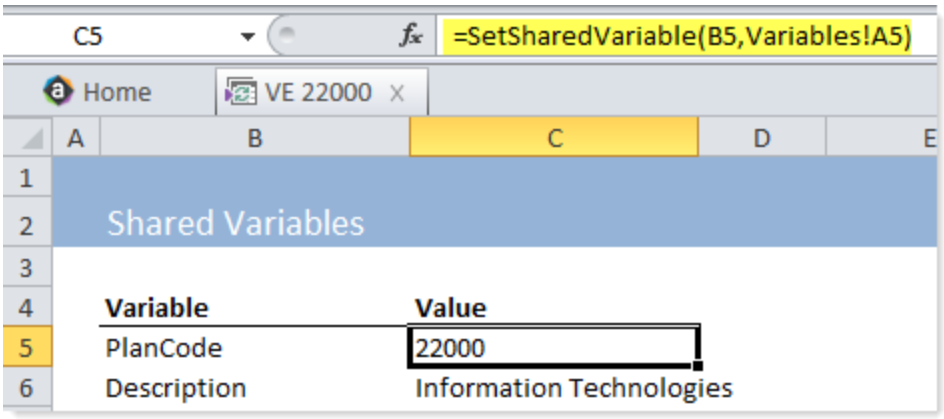
Within the parent template, the current plan code is typically retrieved by using a GetFileGroupProperty data lookup. This data lookup is typically placed on a dedicated sheet (such as "Variables") that retrieves various values used by the template. In the following example, the data lookup result is returned into column D, but we are reading the final values from column A (so that we can use formulas to apply different values for the template versus the resulting plan files).



To share this plan code value with the child utility forms, the parent template must use a SetSharedVariable function that references the result of the GetFileGroupProperty data lookup, such as:

```
=SetSharedVariable("PlanCode",Variables!A5)
```

This function defines the value of a variable named PlanCode, using the result of the GetFileGroupProperty data lookup on the Variables sheet. If the GetFileGroupProperty data lookup returns the value 22000, then the value of the PlanCode variable is 22000.



When the SetSharedVariable function is calculated, the value of the shared variable (as stored in memory for the shared form instance) is set to the result of the function, overwriting any existing value for the variable. In this example the value of the variable is now 22000. The child utility forms can use the GetSharedVariable function to return the current variable value and apply it to data queries, save-to-database processes, and wherever else the value is needed.

For this particular variable example of PlanCode, the value is static. The data lookup that provides the value is only configured to run on open, so the variable value will not change during the current session. But you may have other types of variables where the value could change during the current session, and in that case the variable value will be updated whenever the SetSharedVariable function resolves to a new value.

Keep in mind the following when using the SetSharedVariable function:

- Only one SetSharedVariable function can be used per variable name, in all forms within the shared form instance. Since the value of the variable is set each time the function is calculated, having multiple instances of the function for a single variable name will result in a situation where the last function to be calculated "wins".
- The SetSharedVariable function can be used in either the parent form or any of the child forms, depending on where the variable value is originated. Again, make sure that there is only one SetSharedVariable function per variable name. All other forms in the shared instance should use a GetSharedVariable function to retrieve the value. The GetSharedVariable function can set an initial value for the variable if needed, which will then be overwritten by a SetSharedVariable function if present.

Defining shared variables using interactive components

Interactive components in Axiom forms can be configured to save their selected value to a named shared variable, instead of writing the value directly to the form source file. You should use this approach when the value of the shared variable is set based on the user's selection for an interactive component, such as a combo box, text box, or check box.

One advantage of this approach is that it can provide a two-way street for defining the variable value—where the value can be set in either the parent form or a child form. A two-way street is not possible

when using the SetSharedVariable function, because if the parent and the child forms both have this function, then the variable will always be set to whichever function calculated last. But when using interactive components, the variable value can be set by a component in one form and then read by a component in another form. The current value will be used by both forms until the form user changes the value by interacting with one of the components, in which case both components will then use the new value (though the other form may need to be refreshed before the component is updated to display the new value).

To configure a component to store its interactive value as a shared variable, use the SharedVariable tag. Use of the SharedVariable tag differs slightly depending on what type of interactive component you are configuring:

- For stand-alone components that typically place the interactive value in a component property—such as the Selected Value property for a ComboBox component—the SharedVariable tag is placed in the relevant component property on the Form Control Sheet.
- For content tags that typically place the interactive value in a target cell in the spreadsheet—such as when using the Select tag in a Formatted Grid component—the SharedVariable tag is included as a parameter in the content tag (instead of using the TargetCell parameter).

The syntax for the SharedVariable tag is as follows:

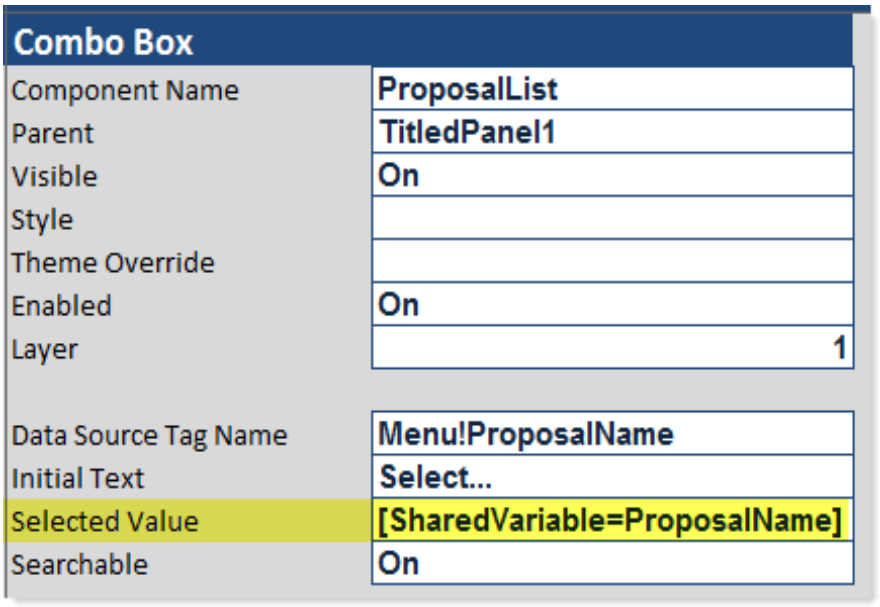
```
[SharedVariable=VariableName]
```

The presence of the SharedVariable tag tells Axiom Software that you want to store the value as a shared variable instead of within the target cell. The VariableName defines the name under which the value will be stored in the shared variables.

For example, if you want the value of a Combo Box component to be stored as a shared variable, you would place the following tag in the **Selected Value** cell for the component:

`[SharedVariable=ProposalName]`

Where the combo box is used to select a Proposal name. If Proposal "Workspace Remodel" is selected from the list, this value becomes the current value for the shared variable ProposalName. GetSharedVariable functions in all of the forms in the shared form instance will return this new value.



Combo Box	
Component Name	ProposalList
Parent	TitledPanel1
Visible	On
Style	
Theme Override	
Enabled	On
Layer	1
Data Source Tag Name	Menu!ProposalName
Initial Text	Select...
Selected Value	[SharedVariable=ProposalName]
Searchable	On

When using the SharedVariable tag, the contents of the Selected Value cell itself are never changed. The selected value is only stored in memory as a shared variable; it does not get written to the file. The only way to reference this value is to use the GetSharedVariable function.

When using this approach, you should also add the variable name to your designated SharedVariables sheet (if using the recommended setup described earlier in this topic), and then use GetSharedVariable function to return the current value of the variable. Any other components or formulas that need to use the selected value of the component should reference this cell on the SharedVariables sheet. You can also use this GetSharedVariable function to set an initial value for the variable, if the component needs to start with a particular value. When the form is first loaded, this initial value will be set as the variable value and therefore displayed as the selected value in the component that uses the variable.

IMPORTANT: Do not use a SetSharedVariable function to set an initial value for the component. If you do this, the SetSharedVariable function will calculate each time the form is updated, and overwrite the current value of the variable back to its original value. Instead, you must use the DefaultValue property of the GetSharedVariable function if you want to set an initial value for the component.

As noted, SharedVariable tags can also be used with content tags in Formatted Grid components. For example, if you want the value for a Select tag to be stored as a shared variable, you would set up the Select tag as follows:

```
[Select; ValueColumn=Proposal.ProposalName; Placeholder=Select a
Proposal; SharedVariable=ProposalName]
```

This Select tag does not contain a TargetCell parameter. Instead, the SharedVariable parameter is used to save the user's selection as the value for a shared variable.

Shared variables can be used with the following components:

Component	Feature or Setting	Notes
Check Box	Is Checked	To set an initial value using the GetSharedVariable function, use True (checked) or False (unchecked). This is also how the function returns the variable value after a user has interacted with the check box.
Combo Box	Selected Value	N/A
Date Picker	Selected Date	• To set a default value using the GetSharedVariable function, use a date string such as "12/31/2018". • When returning a selected value using the GetSharedVariable function, it will be returned as a date/time value with the time set to midnight, regardless of the cell formatting. If you want to use cell formatting to change the display, wrap the function in a DateValue function, such as: `=DateValue(GetSharedVariable ("SelectedDate"))`

Component	Feature or Setting	Notes
Formatted Grid	CheckBox tag DatePicker tag Select tag Selected Row ID TextArea tag	• To set an initial value for the CheckBox tag using the GetSharedVariable function, use 1 (checked) or 0 (unchecked). This is also how the function returns the variable value after a user has interacted with the check box. • To set a default value for the DatePicker tag using the GetSharedVariable function, use a date string such as "12/31/2018". • If the text input for a TextArea tag contains line breaks, then the cell containing the GetSharedVariable function must have Wrap Text enabled in order to display those line breaks when returning the value. For thematic grids, this can be accomplished by applying a column style such as **wrap-text**. • If the TextArea tag is numeric, the number format will be taken from the cell containing the tag. Normally the number format is taken from the target cell, but when using a shared variable there is no target cell. • The notes for the Text Box and Date Picker components also apply when using the TextArea tag and DatePicker tags.
Hierarchy Chart	Selected Value	N/A
Map View	Selected Value	N/A
Pie Chart	Selected Label	N/A

Component	Feature or Setting	Notes
Text Box	Text	• When returning the variable value using the GetSharedVariable function, it is returned as a string value, regardless of the cell formatting. If you want to use cell formatting to display a numeric value, wrap the function in a Value function, such as: `=Value(GetSharedVariable ("RunningTotal"))` This formula will error if the shared variable does not have a value, so you can either use an IF function to handle the no value case, or you can define a default value in the GetSharedVariable function. • If the text box type is Input Mask, and Preserve Input Mask is enabled, then any default value in the GetSharedVariable function must use the input mask format if you want it to display that way. Once the actual value is set, it will automatically use the input mask.
Toggle Switch	Is Checked	To set an initial value using the GetSharedVariable function, use True (checked) or False (unchecked). This is also how the function will return the current state after a user has interacted with the check box.

Returning shared variable values using GetSharedVariable

The GetSharedVariable function returns the current value of a named shared variable. If the variable does not already have a value, the GetSharedVariable function can also be used to set an initial value for the variable.

The GetSharedVariable function takes the following parameters:

```
GetSharedVariable("VariableName","DefaultValue")
```

When the GetSharedVariable function is calculated, Axiom Software checks the list of variables for the shared form instance. If the variable has a value, the function returns that value. If the variable does not currently have a value, the value of the variable is set to whatever is defined in the DefaultValue parameter for the function. That value persists as the variable value until the value is changed by either a SetSharedVariable function or by an interactive component that is configured to use the shared variable. The value can be set by any form in the shared form instance.

For example, imagine the following function:

```
=GetSharedVariable("ReqType","Standard")
```

When the function is calculated, Axiom Software checks the shared list of variables stored in memory. If the variable ReqType currently has a value—for example, "Facilities"—then the function will return that value and the default value in the function is ignored. If the variable ReqType does not currently have a value, then the value is set to "Standard" and the function will return that value. If the function is edited so that the DefaultValue parameter value is now "General", the function will continue to return "Standard" because the variable already has a value. All GetSharedVariable functions in the shared form instance for variable ReqType will return "Standard" until the variable value is changed by a SetSharedVariable function or by an interactive component.

Some examples of how the GetSharedVariable function could be used include:

- A composite form that is a plan file can use multiple embedded forms (utility files) to define the contents of the plan file. The parent plan file form can use SetSharedVariable to define a variable for the current plan code, and then the child forms can use GetSharedVariable to retrieve that value. Similar information, such as the plan code description and other attributes, can also be defined as variables in the parent form and then shared with all child forms.
- A composite form could be structured so that the parent form displays summary financial information, and the child forms are used to develop the detailed financial data. The summary in the parent form could display a running total that is impacted by the current inputs that the user is making in the child form. In order to dynamically update this running total without saving data to the database, the child form could use SetSharedVariable to define a variable for the current running total, and the parent form could use GetSharedVariable to retrieve that value.

When a variable value is set by any form in the shared instance, that value is immediately available to all other forms in the shared instance. There is no need to "push" or "apply" the new value to the other forms. However, in order for the GetSharedVariable function to return the new value, the function must calculate. This means that an update must be triggered for the form in order to reflect any changed variable value. You may need to configure the form as appropriate to force this update to occur. For example, if you are using a Menu component to switch between various child forms, you may need to enable `[ForceRefresh]` for a child form in order to update it for changed variable values. For more information on how form update behavior applies to composite forms, see Form session and update behavior for composite forms.

Using the Apply Shared Variables command

You can use the Apply Shared Variables command to set values for one or more shared variables as needed. Typically this is used to set variables in conjunction with another action.

For example, you may have a grid in an Axiom form where each row shows information about a particular department. You want the user to click a button on the row to open a dialog panel that shows more details about the current department. In order to do this, you need a way to filter the contents of the dialog panel by the department for the current row. You can use an Apply Shared Variables command on the button to set the value of variable Dept to the value for the current row.

Form session and update behavior for composite forms

When using a composite form, the parent and child forms are opened and maintained in a shared form instance. These form sessions are managed and updated using special behavior that is intended to keep all forms in the shared form instance simultaneously active and in sync.

Session behavior

When a user views a composite form, the parent form and all of its child forms are managed together by the Axiom Application Server as a related set of forms (the *shared form instance*). Once a child form has been opened within an Embedded Form component, the session for the child form remains active on the Axiom Application Server until the parent form is closed, even if the child form is not currently visible in the form web page.

For example, imagine a parent form with a Menu component and an Embedded Form component. When the user first opens the parent form, child form 1 is visible as the embedded form. Then the user chooses a different option in the menu, which causes child form 2 to be visible as the embedded form.

When the user switches from showing child form 1 to child form 2, the session for child form 1 is not closed. If child form 1 has unsubmitted values, the form update cycle is processed on the form and the end results of that update are persisted on the Axiom Application Server. If the user switches back to showing child form 1 as the embedded form, the results from the previous update are used to render the form. (The form may or may not be updated again before it is rendered, depending on whether `[ForceUpdate]` is enabled for the target form in the Menu data source.) This behavior allows the form user to access multiple child forms within the shared form instance, and all of those forms will remain active as long as the session for the parent form remains active.

If the form user closes the parent form by any means—such as by closing the browser tab, or by using the browser refresh functionality to reload the form, or by navigating away from the parent form to show a different form within the browser tab—then all of the child form sessions that were related to that parent form are closed too.

Form update behavior when the parent form is initially opened

The following update behavior occurs when the parent form is initially opened:

1. The normal "initial open" behavior occurs for both the parent form and the child form (including running Axiom queries set to refresh on open).
    - The parent form completes its update cycle before the child form begins its update cycle. This allows the child form to access any shared variable values that are set by the parent form.
    - When the child form is updated, the triggering component is logged as the reserved word **$ParentForm**. Normally, the "initial open" process does not have a triggering component, but it is logged in this case to indicate that the parent form caused the child form to open.
2. If the Embedded Form component has **Refresh Parent Form** enabled, then the parent form is

updated again after the child form completes its update cycle. This allows the parent form to access any shared variable values that are set by the child form. If this second update occurs, the triggering component for the parent form is logged as the Embedded Form component.

If Refresh Parent Form is not enabled, then the parent form is not updated again. The previous response for the parent form is used to refresh the web page. If the parent form depends on any values set in the child form, it will not reflect those changes until another update occurs.

3. Once all form updates are complete, the web page is refreshed to show the initial state of the parent and child forms.

This behavior also applies when the parent form is closed and reloaded, such as when using the browser's refresh feature.

Form update behavior when an update is triggered in the parent form

When an update is triggered in the parent form, the following occurs:

1. Before the parent form's update cycle begins, Axiom Software evaluates whether the child form must be updated first. The full form update cycle is processed in the currently visible child form if either of the following apply:
   - The child form has unsubmitted changes.
   - The child form has previously submitted but unsaved changes, and a save-to-database is enabled in the child form.

   Note the following special behavior for this update cycle:
   - The triggering component in the child form is logged as the reserved word **$ParentForm**. This indicates that the update was triggered by the parent form and not by a component in the child form. You can reference this value to enable or disable certain things based on whether the update is triggered by the parent form (such as to enable a save-to-database process or an Axiom query).
   - Axiom Software attempts to execute a save-to-database at the normal part of the child form update cycle if either of the following are true:
     - **Save on Parent Submit** is enabled for the Embedded Form component
     - **Save on Submit** is enabled for the triggering component in the parent

     For more information, see Saving data from composite forms.
   - The end response that updates the child form is fully prepared at the end of the process, but the page itself is not actually refreshed until the parent form has completed its update—at which point the child form may not be visible anymore, such as when the update was triggered by the user clicking on a different item in the Menu component. If so, the response is stored so that it can be used the next time the user switches back to the child form.
2. The full form update cycle is now processed in the parent form. The triggering component for the

parent form is logged as normal.

If the child form was processed first, the parent form now has access to any shared variable values that were set by the child form, and to any data saved by the child form.

3. If the parent form contains a Menu component that determines what is shown in an Embedded Form component, then any additional child form updates depend on the current target of the Embedded Form component (as determined after the parent form update):
    - If the current target is a new child form that is being opened for the first time in the current session, then the normal "initial open" behavior occurs for the child form (including running Axiom queries set to refresh on open).
    - If the current target is the same child form or a previously opened child form, whether the child form is updated again depends on the `[ForceRefresh]` setting in the Menu data source:
        - If True, then the full form update cycle is processed for that child form, allowing it to update for any changes made in the parent form or in other child forms.
        - If False, then the end response that was stored from the child form's last update is used to render the child form.

    If the parent form does not contain a Menu component, and **Force Refresh** is enabled for the Embedded Form component, then the full form update cycle is processed for the child form, allowing it to update for any changes made in the parent form.

    In all cases, if the child form is updated, the triggering component is logged as **$ParentForm**.
4. If the Embedded Form component has **Refresh Parent Form** enabled, and a child form update was processed in step 3, then the full form update cycle is processed again for the parent form. This allows the parent form to access any shared variable values that are set by the new child form.

    If this second parent form update occurs, the triggering component for the update is logged as the Embedded Form component. Keep in mind this means that the triggering component for the second update is different than the triggering component of the first update.
5. Once all form updates are complete, the web page is refreshed to show the current state of the parent and child form.

For example, imagine that the user has made changes in the current child form, but those changes have not yet been submitted. The user then uses the Menu component in the parent form to switch to a different child form. Depending on the refresh settings configured on the Embedded Form component and in the Menu data source, the full update process could be as follows:

- The current child form is updated to preserve its unsubmitted changes before switching to the other child form.
- The parent form is updated to reflect the newly selected menu item (as well as for any other changes to the parent, or for any shared variables set by the current child form).

- The new child form is updated, either for the first time (if initially opened), or to reflect any changed shared variables or new data (if previously opened).
- The parent form is updated again to reflect any shared variables set by the new child form.
- The web page is refreshed to show the current state of the parent form and the new child form.

NOTE: The behavior described here applies when an update is triggered by using a Button component or an interactive component that is set to auto-submit. The Filters panel has special behavior that does not trigger a full form update. For more information, see Defining refresh variables for the Web Client Filters panel.

Form update behavior when an update is triggered in the child form

When an update is triggered in the child form, the following occurs:

1. The full form update cycle is processed in the child form. The triggering component for the child form is logged as normal.
2. If the Embedded Form component has **Refresh Parent Form** enabled, then the full form update cycle is processed in the parent form. This allows the parent form to access any shared variable values that are set by the child form, as well as any data saved by the child form. The triggering component for the parent form is the Embedded Form component. Note the following:
    - The child form is not updated again as part of this process, even if **Save on Parent Submit** is enabled for the Embedded Form component. Because the update process was triggered by the child form instead of the parent form, Save on Parent Submit does not apply in this case. If you want the child form to save data to the database as part of this update, the triggering component in the child form must be configured to Save on Submit.
    - Because the child form was updated first, if the child form depends on any values set in the parent form, it will not reflect those changes until another update is triggered.
3. Once all form updates are complete, the web page is refreshed to show the current state of the parent and child form.

Design considerations

If two or more of the files in the shared form instance have GetData queries with the exact same parameters, and sheet filters are not defined for the relevant sheets, then the GetData queries should be configured to ignore sheet filters. This allows the files to leverage the shared GetData cache for the shared form instance, so that the requested data is only queried from the database once and then used by all requesting files. This applies to both GetData functions and data lookups.

If the sheets with the GetData queries have sheet filters and you want those sheet filters to apply to the GetData queries, then you should not ignore sheet filters, and those queries will be cached on a per sheet basis.

Saving data from composite forms

When using a composite form, there are several options to handle saving data to the database. The method that you choose depends on the flow of data in your forms and on the desired user experience:

- **Automatic Save**: Data is automatically saved from the currently visible child form as the user switches between child forms.
- **Manual Save from Parent**: The user decides when data is saved, by clicking a designated save button in the parent form.
- **Manual Save from Child**: The user decides when data is saved, by clicking a designated save button in each individual child form.

In most cases, data is saved from the child forms only. The parent form is typically just a "frame" that contains the Menu and Embedded Form components. Usually, data is not manipulated or saved from the parent form, though it can be if desired.

Saving data automatically in a composite form

You can configure a composite form so that data is automatically saved in the currently visible child form whenever the parent form is updated. The intended use case for this approach causes the data to be saved as the user switches between forms by using a Menu component. The general user experience is as follows:

- The user makes changes in the currently visible child form (Form 1).
- The user clicks on the Menu component to switch to a different child form (Form 2).
- As part of the update process associated with changing the child form shown in the Embedded Form component, a save-to-database is automatically performed in Form 1 before switching over to Form 2.

This approach ensures that all user changes are saved to the database, and those changes are immediately available to the parent form and other child forms. It is intended for composite form designs that depend on querying changed data from the database, as opposed to sharing unsaved data via shared variables.

One disadvantage of this approach is that users cannot experiment with different data entries and see the effects of their changes without committing them to the database. Also, it may not be clear to users that their inputs are being saved in the absence of an explicit save action.

To set up an automatic save in a composite form:

1. In the parent form, enable **Save on Parent Submit** for the Embedded Form component.
2. In the child forms, set up the save-to-database processes as needed. Note the following:
    - It is required to disable **Save Data Confirmation** in all of the child forms (even if no save-to-database is enabled in a particular child form). The confirmation dialog is not supported when using this configuration, due to the frequency of saving and the timing of showing the confirmation. To disable this setting, click **Edit Form Properties** in the Form Assistant

task pane or Form Designer dialog. (Note that the setting is disabled by default in new forms, so you only have to do this if it was previously enabled.)

- It is *not* necessary to configure a component in the child forms with **Save on Submit**, unless you want to also allow users to trigger a save from within each individual child form. For example, you might want to place a save button on the child forms so that users can save data without needing to switch forms, at which point the automatic save is more like a safety net in case the user forgets to manually save.

When **Save on Parent Submit** is enabled for the parent form, the update behavior works as follows whenever an update is triggered for the parent form (using any component):

- If the currently visible child form has unsubmitted changes, or previously submitted but unsaved changes, the child form performs a full form update that includes executing a save-to-database. (If the child form has no changes, then no action is taken on the child form, and the process skips to updating the parent form.)
- If no save-to-database process is currently enabled in the child form, the save-to-database portion of the update is skipped. No error occurs, and the rest of the form update occurs as normal. Remember that Axiom queries set to "refresh after save data" do not execute if the save-to-database is skipped.
- If an error occurs during the save-to-database process, the child form will complete its update cycle and display the save error, but the parent form will not be updated. This means that if the update is triggered by the user attempting to switch to another child form via the Menu component, the new target form will not load and the Embedded Form component will continue to display the current form. The user cannot move to another child form until the save error is resolved in the current child form.
- If the child save-to-database completes successfully, then the child form update completes. The update process for the parent begins as normal. Other child and parent updates may occur, depending on various settings and whether the target of the Embedded Form component changes. For more information on the composite form update process when the update is triggered by the parent, see Form session and update behavior for composite forms.

Remember that although this method is intended to be used in conjunction with a Menu component, the child update and save-to-database will occur regardless of which component triggers the update in the parent form. For example, if the parent form contains a "refresh" button, or a combo box that is set to auto-submit, then clicking either of these interactive components will also cause the child form to update and save.

Saving data manually from the parent form

You can configure a composite form so that users can save data in the currently visible child form by clicking a designated save button in the parent form. The general user experience is as follows:

- The user makes changes in the currently visible child form (Form 1).
- The user clicks on the Menu component to switch to a different child form (Form 2). The user can then start making changes in Form 2. The changes made to Form 1 are retained on the server but have not yet been saved to the database.
- If a child form has unsaved changes, an asterisk displays on the menu item relating to that form. The asterisk remains until a save-to-database is executed on the child form.
- At any time, the user can click the designated save button in the parent form to update the currently visible child form and execute a save-to-database. In this example, the user would have to navigate back to Form 1 and press the save button in order to save data for Form 1.

This approach is intended for cases where data should only be saved to the database when the user explicitly decides to do so. If the data in one form depends on changes made to another form, and the user needs to be able to see the impact of these changes before saving data, then this data should be passed between forms by use of shared variables. Generally, this is only feasible if a small handful of data points need to be shared. If large amounts of data need to be referenced between forms, it is likely easier to use the automatic save approach instead, so that each form can query the necessary data from the database.

To set up a manual save in a composite form:

1. In the parent form, enable **Save On Submit** for the component that you want to use to trigger the save-to-database. Typically this is a Button component, but it could be any component that supports Save on Submit.

2. In the parent form, set up a save-to-database process. Assuming that you do not need to save data in the parent form itself, this is a "dummy" save-to-database process that only serves the purpose of enabling the Save on Submit component. At least one sheet in the file must have **Save Type 1 Enabled** set to **On** in the main Control Sheet (an enabled Save Type 4 process also qualifies).

   If you do also need to save data in the parent form, then you can set up a real save-to-database process in the parent form. The Save on Submit component will then trigger the save in both the parent and the currently visible child.

   It is required to disable **Save Data Confirmation** in the parent form. If enabled, confusing confirmation messages may result when the save is actually occurring in the child form. To disable this setting, click **Edit Form Properties** in the Form Assistant task pane or Form Designer dialog. (Note that the setting is disabled by default in new forms, so you only have to do this if it was previously enabled.)

3. In the child forms, set up the save-to-database processes as needed. Note the following:
    - It is recommended to disable **Save Data Confirmation** in the child forms (even if no save-to-database is enabled in a particular child form), but not required.
    - It is *not* necessary to configure a component in the child forms with **Save on Submit**, unless you want to also allow users to trigger a save from within each individual child form. However, having a save button in the parent form as well as the child may be confusing to end users. It is recommended to only have a save button on the parent form when using this method.

When a user clicks the Save On Submit component in the parent form, the update behavior is as follows:

- If the currently visible child form has unsubmitted changes, or previously submitted but unsaved changes, the child form performs a full form update that includes executing a save-to-database. (If the child form has no changes, then no action is taken on the child form, and the process skips to updating the parent form.)
- If no save-to-database process is currently enabled in the child form, the save-to-database portion of the update is skipped. No error occurs, and the rest of the form update occurs as normal. Remember that Axiom queries set to "refresh after save data" do not execute if the save-to-database is skipped.
- If an error occurs during the save-to-database process, the child form will complete its update cycle and display the save error, but the parent form will not be updated. Note that when using this method, it is possible for a user to switch to another child form via the Menu component, make changes in that other child form, and save the other child form without error.
- If the child save-to-database completes successfully, then the child form update completes. The update process for the parent begins as normal. For more information on the composite form update process when the update is triggered by the parent, see Form session and update behavior for composite forms.

NOTE: Due to the "dummy" save-to-database process in the parent form, users may see invalid warnings about unsaved changes. For example, if a user opens the form and then uses the Menu component to switch to a child form, that will flag the parent form as having unsaved changes even though the only thing the user has done is use the menu. If the user attempts to close the form in this state, they will see a warning about unsaved changes.

Saving data manually from child forms

If desired, you can configure a composite form so that each child form has its own Save On Submit component that triggers a save-to-database for that child form. The component is typically a Button component, though any component that supports Save On Submit can be used.

You can use this "direct save" approach as the only save method for the composite form, or you can do it in conjunction with the automatic save method discussed previously. It is not recommended to

configure child forms to save independently when using the manual save method on the parent, because the dual save buttons may cause confusion.

When a user triggers an update of the child form using the Save On Submit component in the child form, the full form update cycle is performed on the child form, including a save-to-database. The parent form may or may not be updated as part of this process, depending on whether **Refresh Parent Form** is enabled for the Embedded Form component. If the parent form depends on data that may have been changed by the child form, Refresh Parent Form should be enabled.

If you use this approach, all child forms should be designed so that the Save On Submit component displays in the same place on each form. If different child forms have different configurations, this may be confusing and/or frustrating to the user.

Updating other child forms for saved data

If child forms depend on data that may be changed by saving data in other child forms, you should consider whether `[ForceRefresh]` should be enabled for the child forms. This consideration applies regardless of which save method is used to trigger the save-to-database in the child forms.

Imagine the following scenario:

- When the composite form is opened, the initial child form is Form 1. Form 1 queries data that can be changed using Form 2.
- The user switches to Form 2 via the Menu component, makes changes to Form 2, and triggers a save to database using any method.
- The user switches back to Form 1 via the Menu component. By default, Form 1 is not updated when it is reloaded into the parent form as the current embedded form, because it has been previously opened and has an existing state. This means that the data in Form 1 may be out of date, if it was changed by the save-to-database in Form 2. If you want Form 1 to automatically update to display the changed data from Form 2, then Form 1 must have `[ForceRefresh]` set to True.

The `[ForceRefresh]` option is set in the Menu data source that is used to determine which form displays in the Embedded Form component.

Indicating that forms have unsaved changes

If a child form has unsaved changes, an asterisk displays on the Menu component next to the menu item associated with the child form. This is intended to let the form user know that they need to execute a save on that child form.

This display is necessary when using a manual save method, because users can switch from child form to child form without saving changes. The user must switch back to each unsaved child form and execute a save (either from the parent or the child, depending on how the forms are configured).

When using the automatic save method, only the currently visible child form can ever be flagged as having unsaved changes. It is not possible for the user to switch to another form without saving their changes in the current form first. If the save has errors, the user is prevented from moving to another child form.

Child forms are flagged as having unsaved changes when changes have been submitted back to the source file but no save-to-database has occurred. For example:

- The user selects a value using a ComboBox component in a child form.
- The ComboBox component is set to auto-submit, so the changed value is submitted back to the source file and a form update occurs.
- Axiom Software now detects that the source file has been changed but a save-to-database has not occurred. Keep in mind that Axiom Software doesn't know whether the change actually affects the save-to-database. Any change to the source file will cause it to be flagged as having unsaved changes.
- The menu item that corresponds to the child form now becomes flagged with an asterisk. The asterisk will remain until a save-to-database is executed on the child form.

Keep in mind that because the Menu component is in the parent form, the parent form must be updated in order for the asterisk to show on the menu. Therefore, if an update is triggered in the currently visible child form but Refresh Parent Form is not enabled for the Embedded Form component, the parent form will not update and no asterisk will show on the menu. Once the parent form is updated, the asterisk will show (assuming that the save-to-database still has not occurred).

Hyperlinking to other files in an Axiom form

Within an Axiom form, you can create hyperlinks to other files in the Axiom file system. For example, you may want users to be able to click hyperlinks to open the following kinds of documents:

- Other Axiom forms
- Plan file attachments
- Other non-form Axiom files such as regular reports

NOTE: If you link to a non-form Axiom file, that file must be opened within either the Axiom Excel Client or the Windows Client. Therefore, these types of links are only valid if you expect users to be viewing the Axiom form within the Desktop Client, or within a browser on a machine where the desktop client is installed.

There are three ways that these hyperlinks can be created in an Axiom form:

- Using content tags within a Formatted Grid component
- Using document shortcuts with various form components
- Defining a URL for any component that supports URL

This topic provides a summary of each option and why you might want to use one over the other.

Using content tags in Formatted Grids

You can use the special content tags for Formatted Grid components to create a hyperlink to other Axiom forms, plan file attachments, and other non-form Axiom files.

This format is best used for situations where you want to dynamically generate the list of hyperlinks using an Axiom query. For example, you can query the Axiom.PlanFileAttachments table and automatically generate a hyperlink for each attachment returned by the query.

You can also manually create a static list of hyperlinks within a formatted grid. The only advantage of this approach over the other approaches in this case is that you have more options to control the presentation of the hyperlinks (such as controlling the colors, or using a symbol for the hyperlink instead of display text). However, this approach does require a bit more setup than the other options.

For more information, see Using hyperlinks in Formatted Grids.

Using document shortcuts

You can use document shortcuts within various form components to open a designated file. This is most often used to open a file in the Reports Library (either a regular report or a form-enabled report).

The following components support document shortcuts:

- Button
- Hyperlink
- Sparkline

For Button components you use the Command setting to browse to a file, for the other two components you use the URL setting to browse to a file. If the file is forms-enabled, then you must select the **View As Form** shortcut parameter to open it as an Axiom form. Otherwise the file will open as an Axiom file, requiring use of the Excel Client or the Windows Client.

This approach is appropriate when you have a specific file that you want users to be able to open, and that file is the same for all users of the form.

NOTE: For Button components, the act of opening a file occurs Before Processing and then terminates the normal form update process. The specified file will be opened and no further actions will occur.

Specifying a URL for a component

You can specify a URL for a variety of components in addition to the Hyperlink component, including Image components, and various shape components. This is most often used to link to a web page, but it can also be used to link to Axiom files, if you generate a URL using an Axiom function (such as GetDocumentHyperlink or GetFormDocumentURL). The advantage of this approach is that you can use it on many different components.

Configuring an Axiom form for printing

If a user needs to print an Axiom form, they can do this in one of two ways:

- Print the form using the native print functionality of the browser.
- Use Axiom functionality to generate a PDF of the form and then print the PDF.

If the form is very simple and easily fits within a single standard page, then printing from the browser may be sufficient. However, if the form contains scrolling grids or if you have very specific print requirements, then it is best to configure the form for printing as a PDF. When using this approach you can dynamically adjust the form for the PDF output, to optimize the contents for printing.

There are two ways that users can generate a PDF of an Axiom form for printing:

- In the Web Client toolbar, from the **Tools** menu  , click **Generate PDF**.
- Within the form, use a button that has been specially configured to generate a PDF of the target form.

The button option is recommended because it is more flexible and will work in all environments where forms can be viewed, including in the Desktop Client. Additionally, when using a button you can clearly label the text something like "Print this report" (or similar), which is easier for end users to find and use.

NOTE: If users only need the contents of a Formatted Grid component, not the entire form, then the ability to export grid contents to a spreadsheet may be used as an alternative to printing. For more information, see Exporting Formatted Grid contents to a spreadsheet.

How the PDF is generated

Axiom Software uses a form property named **Is PDF** to track whether the form is currently being used to generate a PDF copy. This system-controlled setting is located at the top of the Form Control Sheet.

When the user initiates PDF creation (using either the Web Client menu or a PDF button within the form), the following occurs:

- On the server, a temporary copy is made of the form source file *as it currently exists* (the "print copy"). If the form has any changes that have not yet been submitted to the source file, these changes will not be reflected in the print copy. In other words, the act of initiating PDF creation does *not* trigger an update of the current form before the temporary print copy is made.
- On the Form Control Sheet of the print copy, the property **Is PDF** is automatically set to **On**, and then the Axiom form is created based on the print copy (with all of the normal refresh behavior that applies when a form is initially rendered). A PDF is generated from the Axiom form, and then the PDF is opened in the browser.

  NOTE: The PDF's own scaling algorithm will be used to fit the content based on the PDF layout settings (see the following section).

Once the PDF is opened in the browser, the user can print and/or save the PDF copy. The specific behavior for printing and saving the PDF depends on the local environment—such as the browser in use and the installed version of Adobe Acrobat Reader.

This approach allows the form designer to dynamically change the form to optimize it for printing. For example, you can show or hide certain components or layers when printing by using an IF formula that references the Is PDF property.

Defining the default page layout for PDF printing

The PDF Size and PDF Orientation settings for the form can be used to set the default page layout for the PDF to the most appropriate configuration for printing. To modify these settings, click **Edit Form Properties** at the top of the Form Assistant task pane or the Form Designer dialog. The settings can also be edited manually at the top of the Form Control Sheet.

- **PDF Size**: Select the desired paper size for the PDF, such as A4, Letter, or Legal. The default size is **Letter**.
- **PDF Orientation**: Select the desired orientation for the PDF: Auto, Portrait, or Landscape. The default orientation is **Auto**, which means that the orientation will be determined based on the width to height ratio of the form (for example, forms that are more wide than tall will be set to landscape).

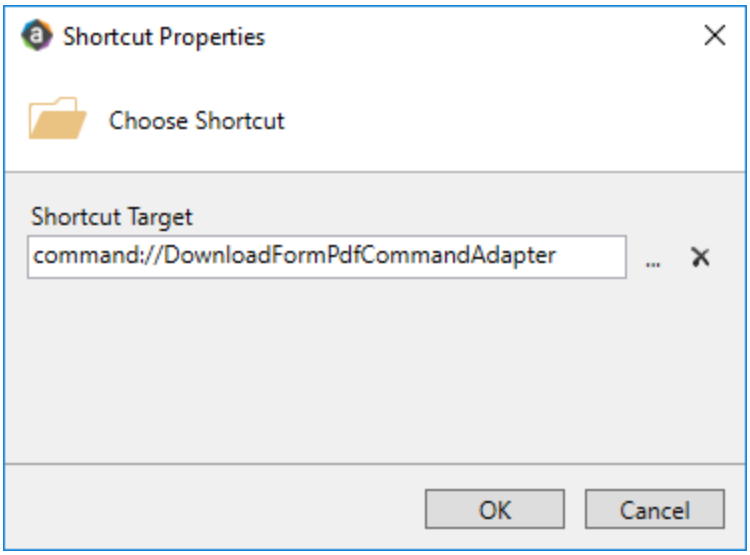
  NOTE: The width and height of the canvas must be explicitly defined in order to calculate this ratio; the assumed canvas size used in the browser is not applied to the PDF. You can use formulas in the **Width** and **Height** settings that reference the **Is PDF** property, so that the width and height can be blank when viewing the form, and then defined when generating a PDF. See Changing the form contents during PDF printing for more information on dynamically changing form settings during PDF generation.

These settings are used to determine the "canvas size" for printing. For example, if the size is set to Letter and the orientation is set to Portrait, then Axiom Software will apply a canvas size of 8.5 x 11 inches to the form.

Generating a PDF using a button in the form

You can place a button on the form itself to allow users to generate a PDF of the current form. The button can be a Button component or a Button tag for a Formatted Grid component.

To do this, use the Command button behavior and then select **Download Form as PDF** as the command. The command does not use any shortcut parameters.



When a user clicks the button, the PDF is generated in the same way as when using **Tools > Generate PDF**.

NOTE: When using this command, the button does not update the current form. Only the PDF generation occurs. This command cannot be combined with other commands.

Changing the form contents during PDF printing

You can dynamically show or hide components or layers in the form for the PDF copy, or change the settings of form components. To do this, use a formula that references the **Is PDF** property on the Form Control Sheet.

For example, in the print copy you may want to hide the print Hyperlink component. You can use a formula in the **Visible** property for that component, such as:

```
=IF(Control_Form!D22="On","Off","On")
```

This formula hides the Hyperlink component when **Is PDF** is On.

If the form contains a Formatted Grid component, this may require special configuration for printing. For example:

- The grid should be configured so that all columns fit the width of the component, otherwise the PDF will not display all columns. This can mean enabling **Fit Columns** for spreadsheet-formatted grids, or configuring sizes in the `[ColumnWidth]` row as appropriate to not exceed the component width (for example, all column widths add up to 100%). If necessary, you can dynamically enable Fit Columns or change the defined column widths based on the Is PDF setting.

- If the number of rows may exceed the height of the component, then **Extended Height** should be enabled for the component, otherwise the PDF will not display all rows. Use of Extended Height will dynamically extend the bottom of the grid to fit all rows, so that they will all display in the PDF. If the extended height behavior should only be applied to the PDF and not when viewing the form online, then you can dynamically enable it based on the Is PDF setting. For more information on using Extended Height, see Setting row sizes for Formatted Grids.
- If the grid uses `[Fixed]` rows, these rows do not have any special treatment in the PDF. They will not repeat at the top of the grid if the grid spans multiple pages.

NOTE: If the target form contains an Embedded Form component to display a child form, Is PDF is also set to On in the child form and can be used to impact the display of the child form. However, the PDF size and orientation properties are taken from the target (parent) form.

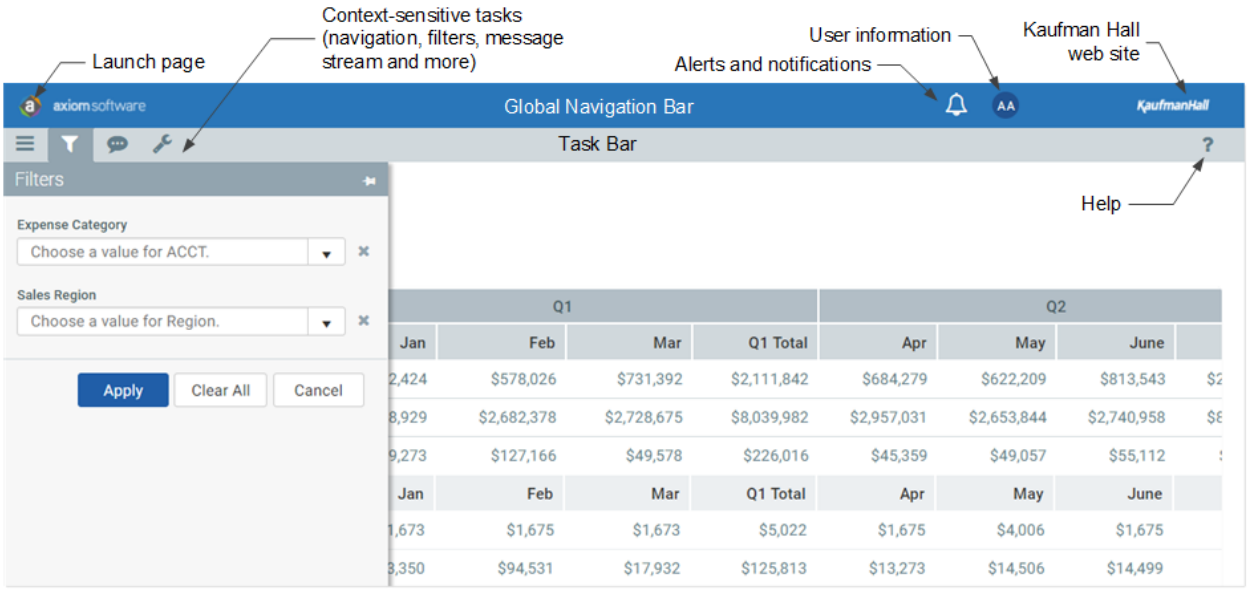
Using the Web Client Container with Axiom forms

When forms are viewed in the Web Client (the browser), they can be shown within a standard "container" that provides access to navigation and other features that can be used on the form. For new forms, the container is enabled by default.

Using the container, form users can:

- Navigate to other web-enabled files or to designated areas of the Web Client
- View alerts and notifications
- View and add comments about the current Axiom form
- Filter the data shown in the current form
- Access tools for use with Axiom forms
- Log out of the Web Client

This container displays as two bars across the top of the form. The top blue bar is the Web Client Global Navigation Bar and the gray bottom bar is the Web Client Task Bar. Clicking an icon in the task bar opens the corresponding feature in a panel along the left-hand side.



Example Web Client Container

Task bar feature overview

Some of the task bar features are entirely built-in, and other features require certain setup steps within the Axiom Software system or within the form itself. The following table summarizes these features, and indicates whether additional setup steps are required:

Feature	Description
Navigation panel	Provides navigation to other web-enabled files and to designated areas of the Web Client. Axiom Software provides a default template for this navigation that can be customized as needed. Additionally, individual forms can optionally have defined navigation links that will display in this panel when the form is open. See Defining navigation links for the Web Client Navigation panel.
Message Stream panel	Displays user comments about the current Axiom form. Users can add new comments. This panel displays the same content as the Message Stream task pane in the Desktop Client. **NOTE:** The message stream is enabled on a per file basis, using the property **Enable Message Stream** on the default Control Sheet. If the message stream is not enabled for the form source file, then the Message Stream panel will not be available for that form in the Web Client.
Filters panel	Allows users to filter the data shown in the current form, by selecting values for defined refresh variables. This must be set up on a per form basis using a RefreshVariables data source. See Defining refresh variables for the Web Client Filters panel.

Feature	Description
Attachments panel	Allows users to upload, open, and manage plan file attachments. This panel is only available for form-enabled plan files, and only if attachments are enabled for the file group. See Using the File Attachments panel to manage attachments.
Tools menu	Opens a menu with access to various form tools, such as the ability to download a copy of the source spreadsheet, create a snapshot copy of the form, or create a PDF of the form.

Enabling the Web Client Container for an Axiom form

The Web Client Container is enabled for new Axiom forms by default. All new forms will automatically have access to this container unless you disable it.

The container, if enabled, only displays in the Web Client (any supported browser) and in the iPad app. It does not display when a form is viewed within the Desktop Client (Excel Client or Windows Client). If the container is enabled but the form is opened in an environment that does not support the container, then the container simply does not display on the form. It is not necessary to disable the setting in order for the form to display properly.

NOTE: The ability to disable the Web Client Container is provided for backward-compatibility only, so that customers with forms created in version 8.2 or earlier can continue to operate "as is" after upgrading. It is intended that these customers will eventually migrate to using the container in conjunction with other enhanced form features.

To enable or disable the Web Client Container for a form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, select or clear the **Use Web Client Container** check box. By default this setting is enabled for new forms.

TIP: You can also enable or disable this setting by editing the **Use Web Client Container** field at the top of the Form Control Sheet.

Generally speaking, the container should be enabled for all forms that will be viewed in the Web Client, or none of them. Having the container appear and disappear depending on the currently open form may be a confusing experience for end users.

Troubleshooting Axiom forms

Several different features are available to help troubleshoot setup issues for Axiom forms.

Error handling for Axiom forms

If an Axiom form has a non-save error, a message will display in the bottom left-hand corner of the form. In some cases you can click on this message to open a more detailed error dialog. Some common non-save errors are as follows:

- If a component that requires a data source does not have an assigned data source, the component will render as a blank component and an error message results.
- If a data source for a component has invalid data, the component will render the valid data only (if possible) and an error message results.

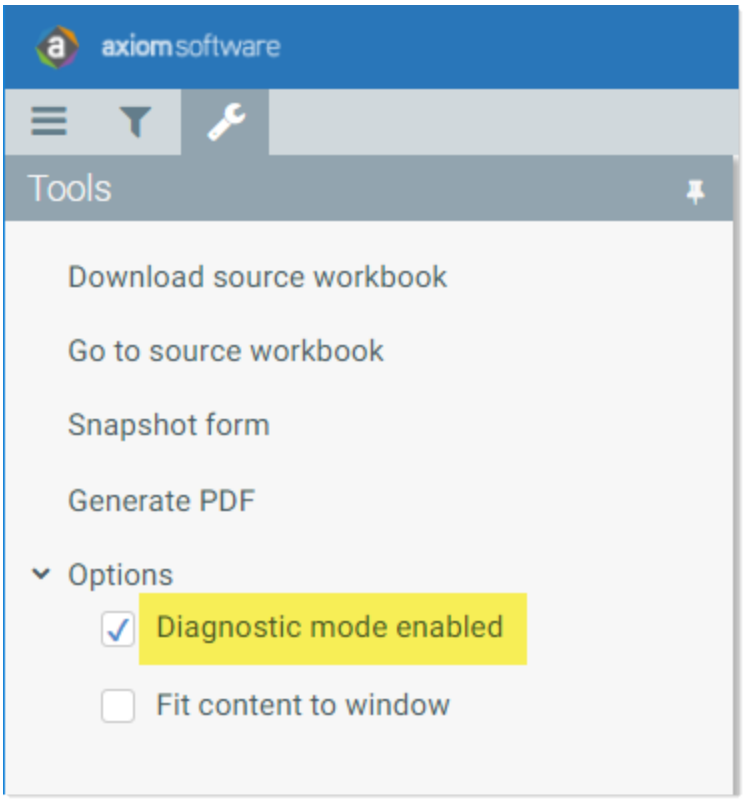
If an error occurs during a save-to-database, this error is displayed in a dialog with information about the error. If it is possible for the user to correct the issue (such as changing an invalid data input) then the user can do so and try the save again.

Form designers should test Axiom forms thoroughly to identify any preventable errors before publishing the form to end users. If there is an error in the form design, there is nothing that the form user can do about the error other than report it to a system administrator.

Using diagnostic mode

When viewing an Axiom form in diagnostic mode, component error messages will display on the affected component in addition to the form-wide error message. This is to help form designers troubleshoot specific components.

Diagnostic mode can be enabled or disabled using the Tools menu in the Web Client task bar. The task bar is only available when the Axiom form is opened in a browser. From the Tools menu (the wrench icon), select or clear the check box for **Diagnostic mode enabled**.



Diagnostic mode is enabled by default for administrators; otherwise it is disabled by default. This setting persists in the current browser session.

Viewing the source file for troubleshooting

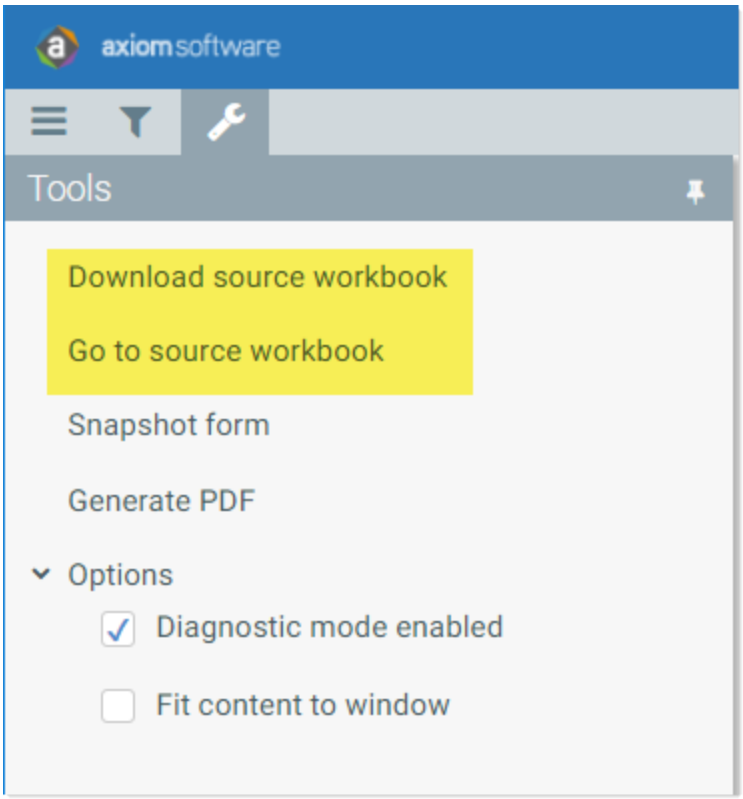
When viewing an Axiom form in the Web Client browser, you have two available options to open the form source file. From the Tools menu (the wrench icon), click either of the following:

- **Download source workbook**: Downloads a copy of the form source file as it currently exists on the Axiom Application Server. Your browser will prompt you to either open or save this file.

  When troubleshooting Axiom forms, it is helpful to be able to see the state of the source file after making changes in the Axiom form. Often this can help illuminate why something is not working as intended. Viewing the updated source file allows you to see exactly what is being submitted back to the source file from the form, and what is occurring in the source file as a response to the update. For example, perhaps an Axiom query is running when you do not expect it to run, and this is overwriting a change that you are trying to make to the source file.

  NOTE: This is a temporary copy for troubleshooting only. If you need to make a change to the source file, you should open the real source file.

- **Goto source workbook**: Opens the actual form source file in the Desktop Client. If the client is not already open, it will be launched.



NOTE: These features are only available to users with read/write access to the file. These features are not supported when using the Web Client on a mobile device.

The **Download source workbook** feature is the only way to see the current state of the file on the Axiom Application Server. If you have the source file open in the Desktop Client, and then preview the form and make changes to the form, you will not see these changes reflected in the file you have open, because that instance of the file is not what is being used to preview the form. In all cases, when you view an Axiom form, a copy of the source file is opened on the Axiom Application Server and that copy is what is being used to render the Axiom form.

Using Form Controls

The form control components provide the basic building blocks of form design and interactivity. In the Form Designer, form control components are available in the **Form Controls** section along the left-hand side of the screen.

- Button: Users can click a button to refresh the Axiom form (including a save-to-database if applicable), and to perform a configured action.
- Check Box: Users can select or clear a check box. The state of the check box is submitted back to the source file.
- Combo Box: Users can select an item from a list. The selected item is submitted back to the source file.
- Date Picker: Users can select a date from a calendar. The selected date is submitted back to the source file.
- Hyperlink: Users can click the hyperlink text to open a web page or a document.
- Image: Display an image such as a company logo.
- Label: Display small amounts of user-defined text, such as for titles, descriptions, or contact information.
- Radio Button: Users can click one of a set of radio buttons to select an option for the Axiom form. The selected button is submitted back to the source file.
- Slider: Users can slide a button along a predefined range to specify a value. The selected value is submitted back to the source file.
- Text Box: Users can type text into the text box. The text is submitted back to the source file.
- Toggle Switch: Users can toggle the switch to Off or On. The state of the switch is submitted back to the source file.

NOTE: Data Grid and Formatted Grid components are also present in the Form Controls area. For more information on these components, see Using Grids.

Button component

The Button component performs an action when a user presses the button. Depending on the button configuration, each button can look and act very differently.

By default, pressing the button triggers the full update cycle of the Axiom form, which can optionally include performing a save-to-database. In some cases, this update may be the only action the button performs. The purpose of the button may be to provide users with a way to refresh the form's data on-demand, or the button may serve as the primary means to submit changes and update the form in response to those changes. If auto-submit is disabled for other interactive components, then the button can be used to trigger the update.

Buttons can also be used to perform other actions in addition to the standard form update. For example:

- Buttons can execute one or more designated commands, such as to run a Scheduler job, or process action codes, or add a row to the source file.
- Buttons can perform specialty actions on plan files, such as to upload a plan file attachment or to complete a process task.
- Buttons can present a multi-select dialog to users, so that they can select one or more items and then those selections will be submitted back to the source file.
- Buttons can open designated in-form dialogs and also perform dialog actions, such as saving data and closing the dialog.
- Multiple Button components can be grouped for use as selection controls, to change something in the form based on which button is currently selected in the group. For example, you could use multiple buttons to show or hide certain layers based on which button is active.

Buttons can be displayed using several different styles. In addition to the default "push-button" display, you can choose to display the button as an image or as a hyperlink. You can also display a symbol on the button in addition to text, or instead of text.

When placing a Button component on the canvas, you can start with the default **Button** or use one of the preconfigured options underneath it. Currently, there are preconfigured options for **Link Button** and **Symbol Button**.

Component properties

You can define the following properties for a Button component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the button. For example, you might want the button to display the text "Refresh". **NOTE:** This setting is ignored if the button style is set to **Image**. You can also optionally omit the button text if a symbol is specified for the button.

Item	Description
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Button Group	Optional. The name of the button group that the button belongs to. You can define a new button group name by typing the name into the box, or you can select from any previously defined group name (within the current form). Button groups only apply when you want to use multiple buttons as selectors. Only one button in the group can be the selected button. For more information, see Using buttons as selectors (button groups). **NOTE:** This setting is only available when using the default button behavior of **Command**.
Is Selected	The current state of the button in the button group, selected or not selected. This setting serves two purposes: • It specifies which button is selected within the button group initially, when the user first opens the Axiom form. By default, this is disabled, which means the button is not selected. If you want this button to be selected initially, enable this setting. • When a user views the Axiom form and selects the button (or selects another button in the group, thereby causing this button to become not selected), the change in the selected status will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the current status of this button. Only one button within a button group can be selected at any one time. **NOTE:** This setting is only available when using the default button behavior of **Command**, and only applies when the button belongs to a button group.
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then clicking the button does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when the button is clicked. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. **NOTE:** This setting is only available when using the button behaviors of **Command** or **Dialog Panel Action**.

Item	Description
Button Behavior	The behavior of the button determines what occurs when the button is pressed. Some button behaviors have additional button properties to configure the behavior. See Button behaviors. By default, the behavior is set to **Command**. This provides the basic button behavior as well as the ability to optionally perform a command. The Command property can be left blank if you do not want to perform a command.
Button Style	The display style of the button. This option is available for all button behaviors except Upload Plan File Attachment. • **Push**: The button displays as a standard rectangular button. The user clicks the button to perform the button action. • **Link**: The button displays as if it is a hyperlink. The user clicks the link to perform the button action. The button text defines the hyperlink text. • **Image**: The button displays as an image. The user clicks on the image to perform the button action. For more information and examples of the different button styles, see Using different button styles.
Symbol	Optional. The symbol to use for the button. The symbol applies as follows: • For push and link buttons, the selected symbol displays on the button in addition to the button text (or instead of the button text, if the text is blank). • For image buttons, you can optionally use a symbol for the button image instead of specifying an image file. To select a symbol, click the **[...]** button to open the **Choose Symbol** dialog. Within this dialog, you can scroll through the available symbols, or you can use the filter box at the top to find symbols based on symbol names. For example, you can type `file` to see all of the symbols that have the word "file" in the name. When you have found the symbol that you want to use, select it and then click **OK**. The selected symbol shows in the Form Designer / Form Assistant, and the actual symbol name is written to the corresponding field in the Form Control sheet. **NOTE:** If you select an image path for an image button, then the Symbol fields are hidden. If you originally selected an image path but now you want to select a symbol, you must first clear the selected image path in order to make the Symbol fields available again. (If you specify both an image path and a symbol by manually editing the Form Control Sheet, the symbol takes precedence.)

Item	Description
Symbol (selected)	The symbol to display when the button is selected. This is intended to be used when the button is part of a **Button Group**, to indicate that the button is the currently selected button in the group (based on the **Is Selected** property). This setting only applies to image buttons, and only if a symbol is specified for the button. The selected symbol is specified in the same way.
Symbol Position	The position of the symbol on the button (**Left** or **Right**). This setting only applies to push and link buttons, and only if a symbol is specified for the button. By default, the symbol displays on the left side of the button. If the text is blank, then this setting does not apply, and the symbol is centered on the button.
Image Path	Specifies the path to the image file to use for the button. This setting only applies if the button style is **Image**. Click the **[...]** button to browse to the image within the Reports Library. If the image is not already saved in the Reports Library, you can right-click a folder and select **Import** to import the image (if you have the appropriate rights to do so). The image must be in PNG or JPG format. **NOTES:** • Users must have permission to the image file in order to see it rendered in the form. It is recommended to create a dedicated Images folder in the Reports Library and store all images in this location. You can grant access to this folder using the Everyone role, or you can create subfolders and grant access to users and roles as needed. • The next time you open this file after saving, the path to the image will be automatically converted into a system-managed document shortcut (you can tell the difference by the presence of a `_tid` parameter on the end of the shortcut). This is to make the file reference "repairable" in cases where the file is renamed or moved. Note that if the path is a result of a formula instead of directly within the cell, then the conversion will not occur and the file reference will not be repairable. You can select either an image path or a symbol for the image button. If you select a symbol, then the Image Path fields are hidden. If you originally selected a symbol but now you want to select an image path, you must first clear the selected symbol in order to make the Image Path fields available again. (If you specify both an image path and a symbol by manually editing the Form Control Sheet, the symbol takes precedence.)

Item	Description
Image Path (selected)	The path to the image file to display when the button is selected. This is intended to be used when the button is part of a **Button Group**, to indicate that the button is the currently selected button in the group (based on the **Is Selected** property). This setting only applies if the button style is **Image** and if an **Image Path** is specified. The selected image is specified in the same way.
Confirmation Message	Optional. Defines a confirmation message to display before performing any button actions. If a confirmation message is defined, then when a user clicks the button in the Axiom form, a message box will display the message. The user can click **OK** to proceed with the button actions, or click **Cancel** to abort the form update and any assigned command. **NOTE:** This setting is only available as described when using the default button behavior of **Command**. The setting is also available when using the **Multi-Select** button behavior, but in that case it defines the header text for the multi-select dialog.
Command	Optional. Specifies a command shortcut to perform when the user clicks the button. Click the **[...]** button to define or edit the command shortcut. Click the ✕ button to clear the shortcut. By default, the button properties only accommodate a single command. You can add multiple commands by clicking the **Add** link underneath the Command box. For more information, see Using buttons to perform commands. **NOTES:** • The command syntax placed in the Control Sheet for the Command setting should not be manually modified or built using a formula. All changes to the command should be made using the Form Assistant or the Form Designer. • This setting only applies when the button behavior is **Command**, Show Dialog Panel, or Dialog Panel Action.

Item	Description
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Button behaviors

When configuring a button, you must specify the button behavior. The default behavior is Command.

General button behaviors

The following behavior options are available in any form-enabled file:

Button Behavior	Description
Command	Updates the form, including optionally performing a save-to-database. The button can also be assigned one or more optional commands. For more information on configuring the button to perform commands, see Using buttons to perform commands.

Button Behavior	Description
Drill	Initiates a drill-down for the currently selected row in a designated Formatted Grid component. This button behavior does not trigger a form update. For more information, see Using a Button component to drill a Formatted Grid.
Multi-Select Items	Opens a built-in dialog where users can select one or more items from a list. Requires a data source to define the list of items. The form is updated when the selected items are submitted. For more information, see Using the multi-select behavior for buttons.
Show Dialog Panel	Opens a designated Dialog Panel component as a modal dialog over the current form. For more information, see Dialog Panel component.
Dialog Panel Action	Performs a designated dialog action on the currently open Dialog Panel component. Available actions are **OK**, **Apply**, or **Cancel**. This behavior only applies when the button is part of a Dialog Panel component; otherwise the button simply triggers a regular form update. For more information, see Dialog Panel component.
Edit Grid Data in Spreadsheet	Opens the contents of a designated Formatted Grid component in a spreadsheet-style editor. Users can edit values and then post changes back to the form grid. This button behavior does not trigger a form update. For more information, see Editing grid contents in a spreadsheet editor.

Planning-related behaviors

The following button behaviors are only available in file group templates and plan files:

Button Behavior	Description
Submit Process / Reject Process	Updates the form (including a save-to-database), and then opens a dialog where the user can complete the active process task for the current plan file. For more information, see Completing the current process task in a form-enabled plan file.
Submit Workflow / Reject Workflow	Updates the form (including a save-to-database), and then opens a dialog where the user can complete the active workflow task for the current plan file. This option relates to the legacy workflow feature.
Upload Plan File Attachment	Opens a dialog where the user can upload an attachment for the current plan file. The form is updated after the upload. For more information, see Using a button in a form to upload attachments.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to

consider the following alternatives:

- Radio Button components can be used instead of Button components to provide button selectors.
- The Button content tag can be used in thematic Formatted Grid components to display a button within a grid. Only the command behavior is supported within the grid, and some limitations apply.
- A Formatted Grid component with CheckBox tags can be used as an alternative to using the Multi-Select Items behavior on a Button component.

Using the multi-select behavior for buttons

Button components can be used to present a selection dialog to users. Users can select one or more items in the dialog, and then those selections will be submitted back to the source file.

To use a button for selection of multiple items, you must:

- Specify the button behavior as **Multi-Select Items**.
- Create a MultiSelect data source in the file to define the list of items, and then configure the button to use that data source.

The multi-select button behavior is best suited for small lists of items. The dialog does not support "type to search" behavior to help users find items in the list. Users must scroll through the list in order to select items.

Configuring a button for multi-select behavior

To configure a Button component as a multi-select button, select **Multi-Select Items** as the **Button Behavior**. Once this behavior is selected, the following additional properties become available for the component:

Item	Description
Data Source	The data source to define the list of items for the selection dialog. You can select from any MultiSelect data source defined in the source file. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Confirmation Message	Optional. Defines explanatory text to display at the top of the selection dialog. If omitted, the dialog will display the text "Select one or more items from the list."

Notice that there is no component property to store the selected items. Instead, items are flagged as selected or not within the MultiSelect data source itself. To configure the form to change based on the selected values, you must reference the `[Selected]` column in the data source.

MultiSelect data source

A MultiSelect data source must be created in the source file to define the list of items. The tags for the data source are as follows:

Primary tag

`[MultiSelect;`*`DataSourceName`*`]`

The DataSourceName identifies this data source so that it can be assigned to a Button component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[ListItem]`

Each row flagged with this tag defines an item to display in the selection dialog.

Column tags

`[Label]`

The display name for each item in the list.

`[Value]`

The corresponding value for each label. This can be the same value as the label, or a different value.

For example, in a list of colors, both the label and the value can be the text Blue. Or, the label text can be Blue while the value is a numeric color code. Separating the label from the value allows you to display "friendly" text to end users but use any value as the selected value.

[Selected]

This column tracks the state of the item, either selected or not selected. Place a `0` (not selected) or `1` (selected) in this column to determine the default state of each item when a user first opens the selection dialog. When a user makes selections and submits them back to the source file, this column will be overwritten with the current state of each item.

NOTES:

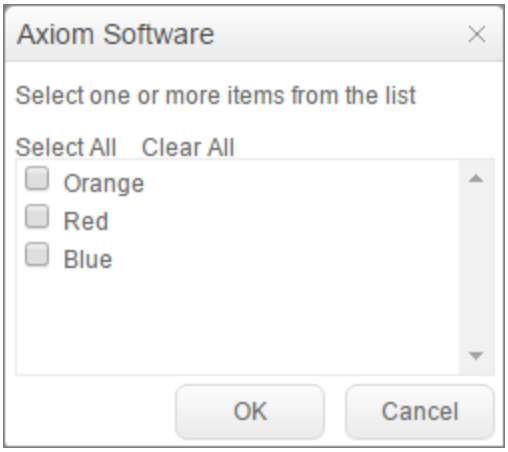
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows sample data flagged in a report. In real implementations this data might be generated by an Axiom query or other query methods; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F
4						
5						
6		[MultiSelect;Colors]	[label]	[value]	[Selected]	
7		[ListItem]	Orange	Orange	0	
8		[ListItem]	Red	Red	0	
9		[ListItem]	Blue	Blue	0	
10						
11						

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > MultiSelect**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight C7:E9) and then use the wizard. Axiom Software will add the tags as displayed in the example above. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting selection dialog appears as follows:



If a user selected both Orange and Red and then clicked **OK**, the `[Selected]` column in the data source would be updated to 1 for those two items.

Multi-select alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- Select tags in Formatted Grid components can be configured to allow multi-select. In this case, the selected items are written to the target cell using a comma-separated list. For more information, see Using drop-down lists in Formatted Grids.
- The Grid refresh variable can be used to allow multi-select in the Web Client filter panel. You may want to do this if the user's selections only impact the data refresh, and do not need to be displayed on the form itself with the other form contents.

It is also possible to use a Formatted Grid component with CheckBox tags to allow users to select multiple items displayed in the grid. The logic employed to determine which items are currently selected would be the same as when using the multi-select button—instead of checking the `[Selected]` column of the MultiSelect data source, you would check the target column of the CheckBox tags. The grid with the check boxes could be displayed directly in the form, or launched using a Dialog Panel as needed. For more information, see Using check boxes in Formatted Grids.

Using buttons to perform commands

The Button component can be used to execute command shortcuts when the button is clicked. This provides a way for users to perform certain actions from within the form.

For buttons, the primary use of command shortcuts is to execute a command from the Command Library. This is the same Command Library that is available for use in custom task panes and ribbon tabs, although only a subset of these commands can be used within Axiom forms.

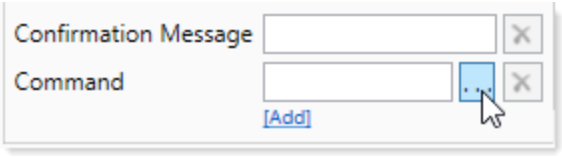
When the user clicks the button in the Axiom form, one or more actions will be performed according to the specified commands. The commands are performed in addition to the standard form update process that is normally triggered by clicking the button. Specific commands may be executed at different points in the update process.

NOTE: Command shortcuts can also be used to open a designated file in the Axiom file system. For more information, see Hyperlinking to other files in an Axiom form. When the shortcut is to a file, the button behaves like a hyperlink and simply opens the specified file, *without* performing the normal update process.

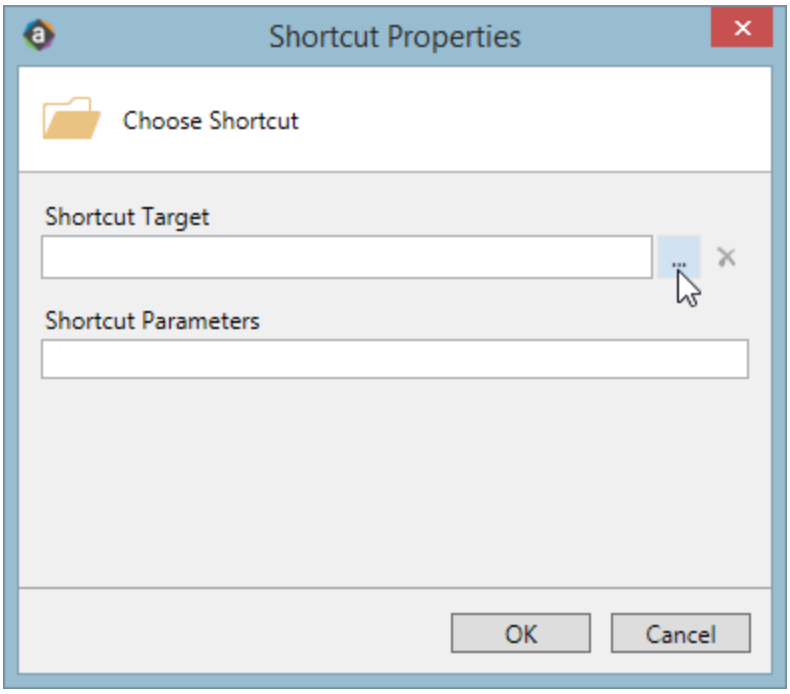
Configuring a Button component to use a command shortcut

This is a generic discussion of how to configure a Button component to use a command shortcut. For more information about how to use specific commands and design considerations, see the individual topics for each command.

1. In the Button component properties (accessible in the Form Designer or in the Form Assistant), click the Browse **[...]** button to the right of the **Command** box.

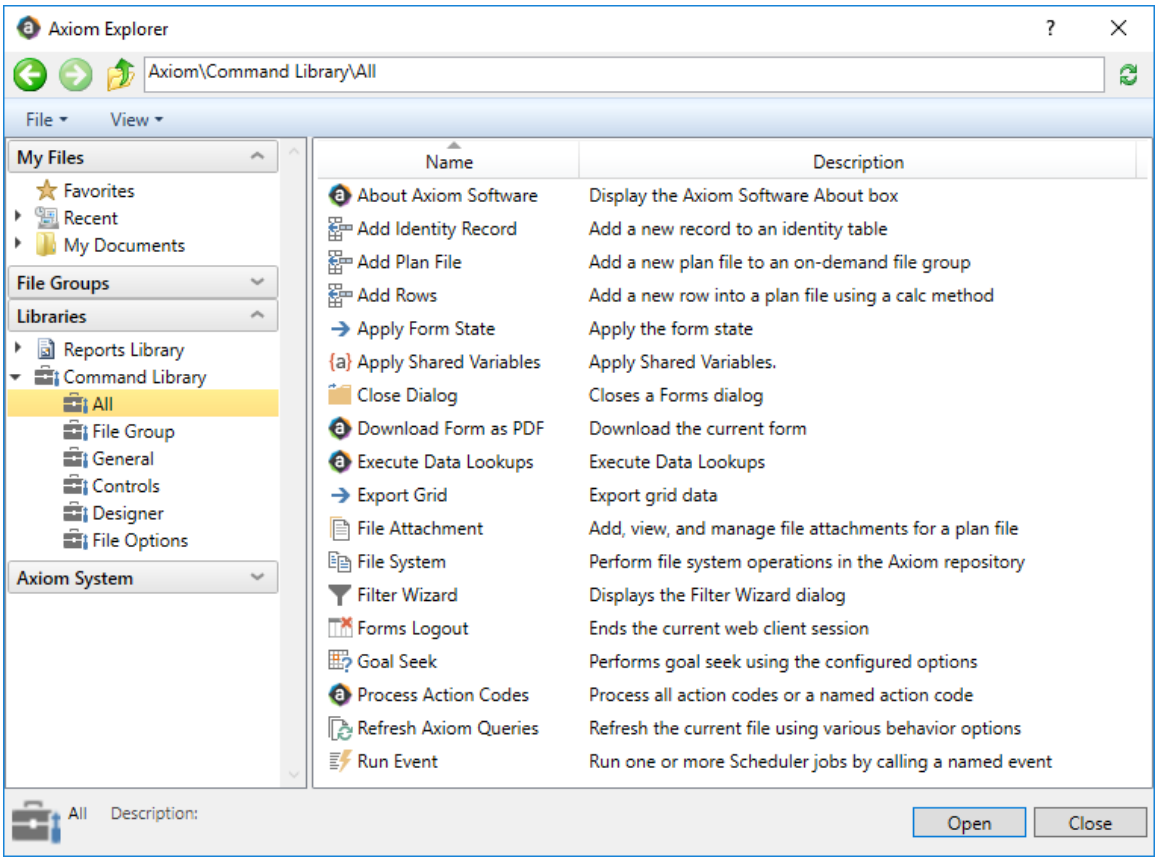


2. In the Shortcut Properties dialog, click the Browse **[...]** button to the right of the **Shortcut Target** box.



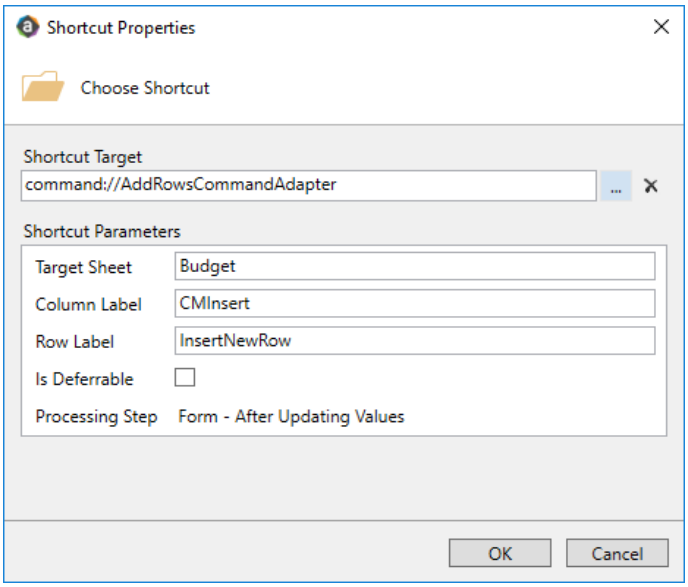
This opens the **Axiom Explorer** dialog, which is filtered to only show items that are eligible for use in command shortcuts within Axiom forms.

3. In the Axiom Explorer dialog, go to the Command Library. Select the desired command shortcut and then click **Open**.



This returns you to the Shortcut Properties dialog, with your selected item listed as the Shortcut Target.

4. If the selected item has **Shortcut Parameters**, configure these parameters as desired.



Example shortcut parameters

The available shortcut parameters depend on the selected command. For more information, see the individual topics on using each command in a form.

In addition to the command-specific parameters, each command has a designated **Processing Step** that determines the timing and context of the command execution. This processing step may be display-only for your information, or it may be configurable. For more information, see Timing of command execution.

5. Click **OK** to close the Shortcut Properties dialog and return to the button properties.

The button is now configured to use the specified command shortcut.

Using a Button tag in a Formatted Grid component

Button tags in thematic Formatted Grid components can also be configured to run commands. In this case, you use the Command parameter within the tag to assign the command to the button. The easiest way to do this is to use the Tag Editor dialog or the Data Source Assistant to create the tag and edit the tag parameters. When using these helper dialogs, you can select the command and configure the shortcut parameters using the same method described previously for the Button component.

Timing of command execution

When a Button component is configured to execute a command from the Command Library, this command is performed during the normal form update process. The command has two properties that determine when and where it is executed:

- The *processing context* determines whether the command is executed on the form (meaning on the form's source file on the application server), or on the active client spreadsheet within the Excel Client or Windows Client. The active client spreadsheet context only applies when the form is being used as a dialog or task pane in the Excel Client or Windows Client.

  Some commands can only be performed in one context or the other, whereas other commands can run in either context.

- The *processing step* determines when the command is executed within the designated context. The available processing steps relate to the form update process that occurs each time a button is used in a form. Some commands are "hard-coded" to only run at a certain processing step, whereas other commands are configurable.

  For example, if the processing context is the form, is the command executed After Updating Values or After Processing? If the processing context is the active client spreadsheet, is the command executed Before Processing or After Processing?

When configuring a command for a button, this information is presented in the shortcut parameters as *processing context — processing step*. If this timing is configurable for a command, you can select the appropriate context / step combination from the **Processing Step** drop-down list.

The following table details the default processing step for each command that can be used in Axiom forms. For more information on the full form update process and the timing of these processing steps, see Axiom form update process.

Command	Processing Step (Default)	Configurable?	Notes
Add Identity Record	Form - After Updating Values	Yes	
Add Plan File	Form - After Processing	Yes	
Add Rows	Form - After Updating Values	No	
Apply Form State	Active Client Spreadsheet - After Processing	No	
Apply Shared Variables	Form - After Updating Values	No	

Command	Processing Step (Default)	Configurable?	Notes
Close Dialog	Active Client Spreadsheet - After Processing	Yes	If the processing step for this command is changed to Before Processing, then this command aborts the form update process. No further actions occur after the command is executed.
Download Form as PDF	Form - Before Processing	No	This command aborts the form update process. No further actions occur after the command is executed. Other commands cannot be combined with this command.
Execute Data Lookups	Form - After Updating Values	Yes	
Export Grid	Form - After Processing	No	Although a form update occurs on the server (for purposes of determining the grid state before export), this update is *not* reflected in the open form. The user will not see any change to the open form.
File Attachment	Form - Before Processing	No	This command aborts the form update process. No further actions occur after the command is executed. Other commands cannot be combined with this command.
File System	Form - After Processing	Yes	
Filter Wizard	Form - After Updating Values	No	This command has special update behavior. See the detailed topic for more information.

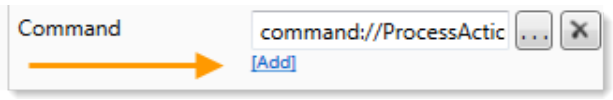
Command	Processing Step (Default)	Configurable?	Notes
Forms Logout	Form - Before Processing	No	This command aborts the form update process. No further actions occur after the command is executed. Other commands cannot be combined with this command.
Goal Seek	Active Client Spreadsheet - After Processing	Yes	
Process Action Codes	Form - After AQ Refresh	Yes	
Refresh Axiom Queries	Active Client Spreadsheet - After Processing	No	
Run Event	Form - After Updating Values	Yes	
Show Form Dialog Panel	Form - After Updating Values	Yes	This command essentially performs the same action as the **Show Dialog Panel** button behavior. For more information, see Dialog Panel component.

NOTE: If the command shortcut points to a file instead of a command in the Command Library, the processing step for opening that file is effectively Before Processing. The file will be opened and then no further actions occur.

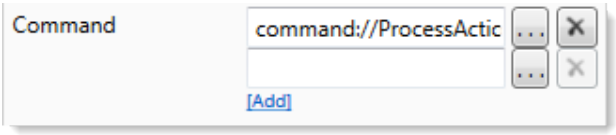
Using multiple commands on a button

Button components can perform multiple commands. By default, the button properties only have one command box, but you can add more command boxes to accommodate multiple commands.

To add more commands to a button, click the **Add** link below the **Command** box in either the Form Assistant or the Form Designer. This will add another command box to the component, where you can configure another command. You can add as many commands as necessary.



Click Add to add more commands



New blank box becomes available for additional command

NOTE: Multiple commands cannot be used if the command shortcut opens a file instead of executing a command from the Command Library. When the command shortcut points to a file, the file is opened when the button is pressed, and no further actions occur.

If a button uses multiple commands, then each command is performed according to its configured or built-in processing step (see the previous section Timing of command execution for more details). For example, a button could have two commands: Process Action Codes and Run Event. The Process Action Codes command could be configured to run After Updating Values, and the RunEvent command could be configured to run After Processing. Both commands would be performed during the normal update process, at their assigned points in the process.

If each command is performed at a different processing step, then the order of the commands within the button properties is irrelevant. However, if multiple commands will be performed at the same processing step—for example if two commands will both be processed After Updating Values—then the commands will be performed in the order they appear in the button properties.

NOTE: If the Close Dialog command is configured to execute at **Active Client Spreadsheet - After Processing**, then the order of the command in relation to other commands being performed at the same processing step is irrelevant. Other commands that are configured to execute at that processing step will still execute after the dialog is closed.

Currently, there is no way to reorder commands for a Button component using the Form Assistant or the Form Designer. If you need to reorder commands, you can go to the Form Control Sheet and do one of the following:

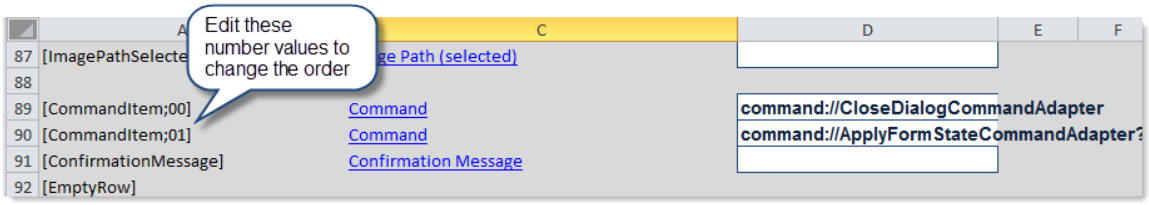
- Swap the contents of the Command boxes for the commands that you want to reorder. For example, copy and paste the contents of the top command to a temporary location. Then copy and paste the contents of the bottom command to the top command. Then go back the temporary location and copy and paste the original contents of the top command to the bottom command.

- Alternatively, you can manually change the order number of the commands as detailed in column A of the Form Control Sheet. The command tags in column A use the following syntax:

  `[CommandItem;XX]`

  Where XX is a number from 00 to 99. The numbers are relative to the current component only.

  To do this, clear the freeze panes for the Form Control Sheet and then move to column A. Locate the command tags and then change the numbers as desired. For example, if the first command is 00 and the second command is 01, swap the numbers for each command.



Using buttons as selectors (button groups)

You can use multiple Button components as selectors in a form. To do this, you assign two or more buttons to a designated button group. You can configure the form to change based on which button is the currently selected button in the group. When using this approach, the Button components behave like Radio Button components.

Using buttons vs radio buttons

There are a couple of reasons why you might want to use regular buttons instead of radio buttons as selectors:

- Button components can display as images or as hyperlinks in the form. If you want users to select by clicking images or links instead of radio buttons, then you must use Button components.
- Button components can execute commands when they are clicked. If you want to execute a command when a particular button is clicked, then you must use Button component.

One disadvantage of using Button components instead of Radio Button components is that Button components do not have a built-in visual representation to indicate which Button component is the currently selected button. There is no "pressed" style for Button components. However, if you use the Image button style, you can assign a different image to display when the button is selected.

Configuring Button components as selectors

In order to use Button components as selectors, you must configure the buttons as follows:

- The **Button Behavior** property must be set to **Command**. You do not need to use a command, but this is the only behavior setting that supports button groups.
- The **Button Group** property must be set to the name of the button group. You can type in a name to create a new button group for the form, or you can select from button groups that have already been created. Only one button in a button group can be the currently selected button.
- If you want one of the buttons in the button group to be selected by default, then enable the **Is Selected** check box for that button. When a user opens the form, this button will be selected to start. If the user then clicks a different button in the button group, that button will be marked as selected and the original button will be cleared.

As previously noted, it is recommended to use the Image button style when using Button components as selectors. You can have two versions of each button image—an "unselected" image and a "selected" image. For example, the unselected image could show an icon, and the selected image could show a color-reversed version of the same icon. These images should be specified as the **Image Path** and the **Image Path (selected)** respectively. When a user clicks on a button to select that button, the button will update to display the "selected" image. (Alternatively, symbols can be used instead of image files.)

Using button groups

Button groups track the "currently selected" state among two or more buttons. When a user clicks a button that belongs to a button group, the **Is Selected** setting is enabled for the button that was clicked, and disabled for all other buttons in the group. This allows you to configure interactivity for the form based on the currently selected button in the group, such as hiding or showing certain content in the form.

What occurs when a user selects a radio button is entirely up to the form designer. For example, a form could have three image buttons that show images representing the three main product lines of your organization. These three buttons could belong to a button group named Product. The form could also contain other components such as grids and charts that show data related to the currently selected product line. These other components would be set up using formulas to change the data displayed based on the currently selected button in the Product button group.

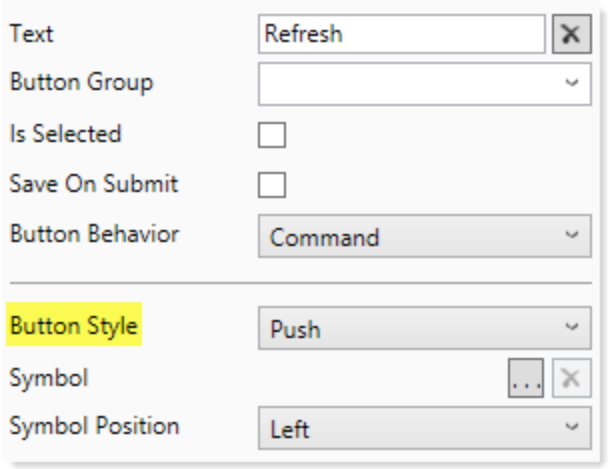
If a Button component belongs to a button group, the button still triggers a form update and performs any commands it is configured to perform. The only difference is that the **Is Selected** field will also be updated for the button and all other buttons in the group, which may in turn impact other components or data queries in the form that have been configured to reference this setting.

Using different button styles

Button components can be presented using three different button styles:

- **Push**: The button displays as a standard rectangular button. The user clicks the button to perform the button action.
- **Link**: The button displays as if it is a hyperlink. The user clicks the link to perform the button action. The button text defines the hyperlink text.
- **Image**: The button displays as an image. The user clicks on the image to perform the button action.

The button style is specified using the **Button Style** property, as shown in the following screenshot:



By default, all buttons are push buttons unless the button style is changed. Some of the preconfigured button alternatives start with a different button style—for example, the **Link Button** alternative is automatically set to use the link button style.

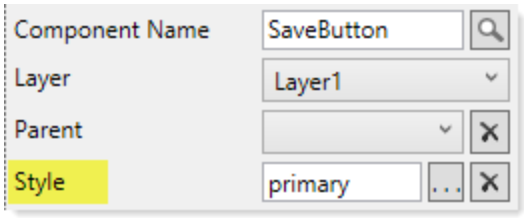
The choice of which button style to use is primarily a design decision. The button style does not impact the button functionality; it only impacts how the button displays to form users.

All button styles can also use symbols from the symbol library. These are the same symbols that are available when using the Symbol tag in a Formatted Grid component.

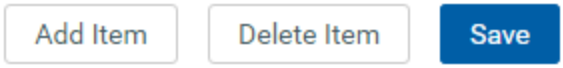
Push buttons

Push buttons use the standard button appearance that you are used to seeing in software dialogs and web pages. Users click the button to perform an action.

Push buttons can use the **primary** style to designate a button as the primary action for the form. For example, if you have a row of buttons that perform actions, but you want to call attention to the Save button, you can apply the primary style to that button using the **Style** property.

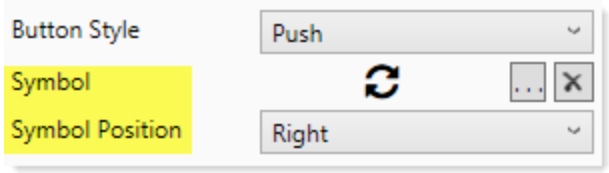


Component properties with primary style applied



Push button using primary style

The text on a push button can include a symbol. To apply a symbol, click the **[...]** button for the **Symbol** property to select a symbol, and then specify whether the symbol should display on the left or right side of the button.



Component properties with symbol selected

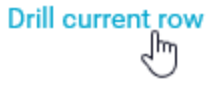
Push button using symbol with right position

If you want the button to only contain a symbol, then you can leave the text blank. In this case the symbol position does not apply, and the symbol is centered in the button.

Push button using only a symbol

Link buttons

Link buttons display as hyperlink text. Instead of clicking a button to perform an action, users click the hyperlink text.

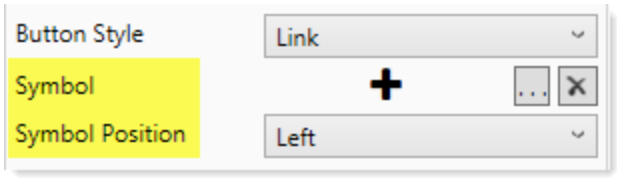


By default, link buttons display in blue font and without an underline. If desired, you can apply the following styles to the button to impact the display:

- **text-color**: Removes the blue color and instead displays in the default text color.
- **underlined**: Applies an underline to the text.

You can also use the formatting overrides in the advanced component settings to change the font properties.

The text on a link button can include a symbol. To apply a symbol, click the **[...]** button for the **Symbol** property to select a symbol, and then specify whether the symbol should display on the left or right side of the button.



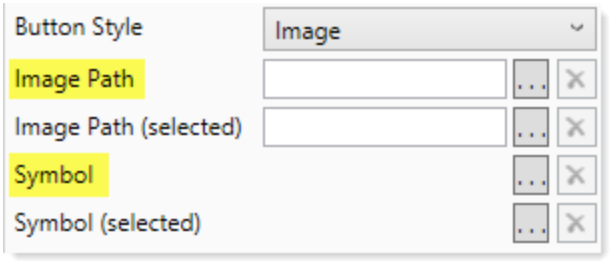
Component properties with symbol selected

Link button using symbol with left position

Image buttons

Image buttons display as an image. Instead of clicking a button to perform an action, users click the image.

When configuring an image button, you must specify either an **Image Path** or a **Symbol**. If an image is selected, then the symbol fields will become hidden, and vice versa. Click the applicable **[...]** button to browse to the desired image, or to select the desired symbol.



If the button is part of a button group, then you can optionally specify a "selected" image or symbol. When the button is the currently selected button in the button group, then the button will use the selected image / symbol instead of the primary image / symbol. For example, this button will use the regular star when it is not selected, and switch to the filled star when it is selected.

Check Box component

The Check Box component is an interactive component that displays a check box on the Axiom form. Users can select or clear the check box to enable or disable something in the form.

Component properties

You can define the following properties for a Check Box component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the check box.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.

Item	Description
Is Checked	The current state of the check box, checked or not checked. This setting serves two purposes: • It specifies the initial state of the check box, when the user first opens the form. By default, this is disabled, which means the check box is not selected. If you want the check box to be selected initially, then enable this setting. • When a user views the form and selects or clears the check box, this state change will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the current state of the check box. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the checked status to be read from and written to the specified cell reference instead of directly within the Is Checked cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the checked status is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Auto Submit	Specifies whether the Axiom form is automatically updated when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user selects or clears the check box. If this setting is disabled, then the user must use a Button component in order to update the form for the changed state. For example, you might disable the auto-submit behavior if the check box is one of several user selections that are intended to be submitted together at one time, instead of piecemeal as each one changes. In that situation the user can make all necessary changes for all related components, and then click a Button component to submit the changes at once and trigger an update.

Item	Description
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Check Box components. Only the generic styles are available. Most check box styling is controlled by the form-level skin. The text for the check box can be formatted using the advanced settings.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Check Box component allows the user to check and uncheck a box. The current state of the check box is submitted back to the source file, and written to the **Is Checked** setting on the Form Control Sheet.

If you want the Axiom form to respond to the state of the check box (on or off), then you must set up the file so that another component references the check box state and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

An Axiom form could contain a column chart with three possible series. By default the chart shows two series, but if the check box is selected, the third series is shown. The series tag for the third series could be set up using an IF function so that the series only displays if the check box is selected. For example:

```
=IF(Control_Form!D298="Off","","[Series]")
```

In this example, the Is Checked setting for the check box is located on the Form Control Sheet in cell D298. Therefore if the check box state is Off, then this cell will be blank and the series of data in this row will not display in the chart. But if the check box state is On, then the series tag will display and therefore the associated data will display in the chart.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- The CheckBox content tag can be used in Formatted Grid components to present interactive check boxes within a grid. You may want to do this if your form is primarily grid-based, or if the check boxes need to be integrated with the other contents of the grid (such as displaying a check box on each row of a grid). For more information, see Using check boxes in Formatted Grids.

Combo Box component

The Combo Box component is an interactive component that displays a drop-down list of items in the Axiom form. Users can select an item from the list to impact the form in some way.

Defining a combo box is a two-part process that requires the following:

- Creation of a ComboBox data source in the spreadsheet to define the list of items.
- Placement and configuration of a Combo Box component on the Axiom form canvas.

Data source tags

A Combo Box component must have a defined data source in the file to define the list of items. There are two different ways to define this data source:

- You can use the default approach of defining the list of items within the sheet using column and row tags. This approach is described below.
- Alternatively, you can use "extended syntax" for the ComboBox tag to define the list of items by referencing a table column or an Axiom query. This works in a similar manner as the Select tag for Formatted Grid components. For more information on this approach, see the following sections:
  - Using a table column as the source for a combo box
  - Using an Axiom query as the source for a combo box

The tags for the default Combo Box data source are as follows:

Primary tag

`[ComboBox;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a Combo Box component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[ComboItem]`

Each row flagged with this tag defines an item to display in the combo box.

Column tags

`[Label]`

The display name for each item in the list. Labels should be unique. If multiple rows have the same label, then the first value with that label is used.

[Value]

The corresponding value for each label. This can be the same value as the label, or a different value.

For example, in a list of colors, both the label and the value can be the text Blue. Or, the label text can be Blue while the value is a numeric color code. Separating the label from the value allows you to display "friendly" text to end users but use any value as the selected value.

NOTES:

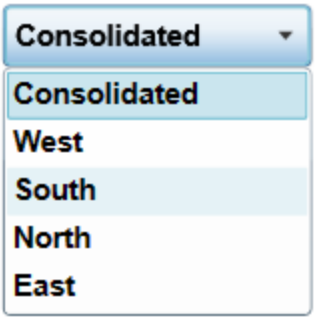
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows sample data flagged in a report. In real implementations this data might be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D
1				
2		[ComboBox;Regions]	[Label]	[Value]
3		[ComboItem]	Consolidated	All
4		[ComboItem]	West	West
5		[ComboItem]	North	North
6		[ComboItem]	South	South
7		[ComboItem]	East	East

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Combo Box**. You can also highlight a range of cells first and then use the wizard (in the example above, you would highlight C3:D7). Axiom Software will add the tags as displayed in the example above. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The resulting combo box for this example data source would appear as follows:



If a user selects the label Consolidated from the combo box, the corresponding value of All is written to the **Selected Value** property of the data source.

Component properties

You can define the following properties for a Combo Box component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the combo box, to determine the list of items in the box. A data source must be tagged within the file as specified above, and then selected for the combo box. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Initial Text	The text to display on the button if the Selected Value is blank. By default, this text is "Select..." to prompt the user to select a value from the drop-down list. If you want to use different text, enter that text into this field. Blank initial text is not supported. If this setting and the Selected Value setting are both blank, then the combo box will display "Select..." The appearance of the initial text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the initial text displays in a lighter color than selected values, but not always.

Item	Description
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Selected Value	The currently selected value for the combo box. This setting serves two purposes: • It defines the initially selected value for the combo box, when the user first opens the form. By default this setting is blank, which means that no value is selected for the combo box, and instead the combo box will display the text defined for the Initial Text setting. Alternatively you can enter a value from the Values column of the data source, and the combo box will display that value to start. • When a user views the form and selects an item from the drop-down list, this selected value will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected value for the combo box. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected value to be read from and written to the specified cell reference instead of directly within the Selected Value cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected value is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Searchable	Specifies whether the combo box includes the ability to type in values to search the list. By default this is disabled, which means that the user can only select values from the list. The user cannot type in a value to find a matching value. If enabled, then the user can click into the box to start typing text. This filters the list for matching values, which the user can then select. Typically this setting is only enabled for longer lists where it would be inconvenient for the user to have to scroll to the value, or where the total list exceeds the number of items that can be shown (in this case the user can still search to select the value).

Item	Description
Auto Submit	Specifies whether the Axiom form is automatically updated when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user selects an item from the combo box. If this setting is disabled, then the user must use a Button component in order to update the form for the changed state. For example, you might disable the auto-submit behavior if the combo box is one of several user selections that are intended to be submitted together at one time, instead of piecemeal as each one changes. In that situation the user can make all necessary changes for all related components, and then click a Button component to submit the changes at once and trigger an update.
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Combo Box components. Only the generic styles are available. Most combo box styling is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

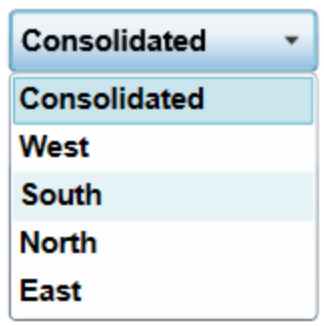
The Combo Box component allows the user to select an item from a list. The selected item is submitted back to the source file, and written to the **Selected Value** setting on the Form Control Sheet, using the value in the `[Value]` column of the data source (not the display name in the `[Label]` column).

If you want the Axiom form to respond to the currently selected item, then you must set up the file so that another component references the selected item and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

An Axiom form could contain a column chart that dynamically changes its series or its data based on a combo box. For example, the combo box could allow the user to select a region, or a company, or an account type, and then the chart would update to show the selected item.

Example

A combo box could display values such as the following:



In this case the default value for the combo box (the value saved in the Selected Value setting for the file) is Consolidated. The user can change this value by selecting a region, such as West. This causes the value West to overwrite the existing value in the Selected Value setting. A column chart could look to this setting to determine what to display in the chart.

There are a variety of ways that the chart could update based on this setting. For example, the data filter for the Axiom query could change based on the setting, so that the data coming into the sheet is different depending on what the user selects. Or, all of the data could already exist in the sheet, and the chart could simply show or hide particular data series based on the user's selection (by setting up the series tags using a dynamic formula).

Using a table column as the source for a combo box

The Combo Box component can use the key column of a reference table as the source for the list. For example, you might want to display a list of accounts in a combo box.

To do this, use the following "extended syntax" for the ComboBox tag. In this case, the ComboBox data source only consists of this tag; there are no column and row tags.

```
[ComboBox; DataSourceName; ValueColumn=ColumnName;
DescriptionColumn=ColumnName; SortColumn=ColumnName; DisplayCell=CellAddress;
DisplayFormat=Text; Filter=FilterStatement]
```

Parameter	Description
ValueColumn	The column to provide the list of values for the combo box. Enter a fully-qualified Table.Column name such as `Acct.Acct`. Multi-level lookups can be used. You can specify any column from any customer-defined table in your system. System tables such as Axiom.Aliases are not supported for use with refresh variables and cannot be used. When using columns with lookups (including multi-level lookups), the final lookup table is considered the primary table. For example, if you specify `GL2018.Dept`, this is the same as specifying `GL2018.Dept.Dept`, so the Dept table is the primary table. Any columns listed in filters and as additional columns must be resolvable from the primary table, or must contain a fully qualified path from the starting table (GL2018 in this example). When using columns with lookups, the starting table impacts the list of items to be returned from the value column. For example, `GL2018.Dept` returns only the departments used in the GL2018 table, whereas `Dept.Dept` returns the full list of departments defined in the Dept table.
DescriptionColumn	Optional. The column that contains descriptions for the value column, specified using a fully qualified Table.Column name or an alias name. For example: Acct.Description. This property only applies if the value column is a key column or a validated column. By default, the primary table's first description column will be displayed in the list if no alternate description column is specified in the tag. You only need to complete the DescriptionColumn parameter if the table has more than one description column and you want to specify a different description column. However, if you are using the DisplayFormat parameter to define a custom display format, then the DescriptionColumn parameter does not apply. Instead, you should include the description column in the custom display format as desired.
SortColumn	Optional. The column by which to sort the list of values. By default, the list is sorted by the display format if defined, and by the value column if no display format is defined. You can use this property to override the default sort and instead specify a different column to sort by. If the value column uses a lookup, then the column must be resolvable from the primary table, or must use a fully qualified path from the starting table.

Parameter	Description
DisplayCell	Optional. The cell that contains content that you want to display in the combo box other than the selected value. Specify either a cell reference (A22) or a column letter (A). If only a column letter is specified, the current row is assumed. For example, if the purpose of the combo box is to select an account, and the account list displays items as Account - Description, then you may want the selected value to also display as Account - Description instead of just Account. You could have a formula in the designated display cell that creates this alternate display based on the selected value.
DisplayFormat	Optional. Defines a display format for the items in the list, and specifies additional columns to display. By default, items in the list are displayed as: *KeyColumn - DescriptionColumn* If you want to specify a different format and/or use additional columns, then you can indicate the display format here. Use fully qualified Table.Column syntax and place column references in curly brackets. For example, you could indicate something like: {Acct.Acct} - {Acct.Description} ({Acct.Category}) This would display account items in the following format: 8000 - Facilities (Overhead) Any columns listed should use fully qualified Table.Column syntax. If the value column uses a lookup, then any additional columns must be resolvable from the primary table, or must use a fully qualified path from the starting table. Additional columns included in the display format are searchable within the list.
Filter	Optional. A filter criteria statement to limit the values available for selection. The filter impacts both what displays in the drop-down list, and what is available when searching using the filter box. If the value column uses a lookup, then the column in the filter criteria statement must be resolvable from the primary table, or must use a fully qualified path from the starting table.

Parameters can be listed in any order after the ComboBox tag and the data source name. You do *not* need to indicate omitted parameters with an "empty" semi-colon.

To create the extended ComboBox tag, you can manually type it within a cell, or you can use the Combo Box Tag Helper to assist you in creating a tag. To open the helper dialog, place the basic ComboBox tag within a cell—meaning `[ComboBox;ComboBoxName]`—then double-click the cell.

Note the following when using the extended ComboBox tag:

- The selected value is still placed in the **Selected Value** cell of the component properties. Indirect cell references and form state / shared variable tags can be used as normal.
- In most cases, **Searchable** should be enabled in the component properties so that users can search the list. All columns displayed in the list are included in the search. However, if the list only has a small number of items then this may not be necessary.

  NOTE: Only the first 100 values from the data source are displayed in the drop-down list. If the list contains more than 100 values, then Searchable must be enabled so that users can search for the other values. This limit does not apply when using the regular `[ComboItem]` syntax; the limit only applies to extended tags.

- In the form, the **Initial Text** will show in the combo box until a value is selected. Once a value is selected, the contents of the Selected Value field will show unless a DisplayCell has been specified, in which case the contents of the DisplayCell will show.

Using an Axiom query as the source for a combo box

The Combo Box component can use an Axiom query as the source for the list.

To do this, use the following "extended syntax" for the ComboBox tag. In this case, the ComboBox data source only consists of this tag; there are no column and row tags.

`[ComboBox; DataSourceName; AQ=Sheet!AQName; ValueColumn=ColumnName; DescriptionColumn=ColumnName; DisplayCell=CellAddress; DisplayFormat=Text; Filter=FilterStatement]`

Parameter	Description
AQ	The name of the Axiom query to use as the source of the list. This parameter uses the following syntax: *SheetName*!*AQName*. For example: Sheet2!AQList. The setup requirements for the Axiom query are the same requirements used by the Select tag for Formatted Grid components. For more information, see Axiom query setup for use in a Select tag.
ValueColumn	Optional. The column that contains the list of values, specified using a fully qualified Table.Column name or an alias name. For example: Acct.Acct. By default, the first column in the Axiom query field definition is assumed as the value column. You only need to specify the value column in the tag if it is not the first entry in the field definition.

Parameter	Description
DescriptionColumn	Optional. The column that contains descriptions for the value column, specified using a fully qualified Table.Column name or an alias name. For example: Acct.Description. By default, the second column in the Axiom query field definition is assumed as the description column. You only need to specify the description column in the tag if it is not the second entry in the field definition and if you are not using the DisplayFormat parameter to define a custom display format. However, if you are using the DisplayFormat parameter to define a custom display format, then the DescriptionColumn parameter does not apply. Instead, you should include the description column in the custom display format as desired.
DisplayCell	Optional. The cell that contains content that you want to display in the combo box other than the selected value. Specify either a cell reference (A22) or a column letter (A). If only a column letter is specified, the current row is assumed. For example, if the purpose of the combo box is to select an account, and the account list displays items as Account - Description, then you may want the selected value to also display as Account - Description instead of just Account. You could have a formula in the designated display cell that creates this alternate display based on the selected value.
DisplayFormat	Optional. Defines a display format for the items in the list, and specifies additional columns to display. By default, items in the list are displayed as: *KeyColumn - DescriptionColumn* If you want to specify a different format and/or use additional columns, then you can indicate the display format here. Use fully qualified Table.Column syntax and place column references in curly brackets. For example, you could indicate something like: {Acct.Acct} - {Acct.Description} ({Acct.Category}) This would display account items in the following format: 8000 - Facilities (Overhead) Any column used in the display format must also be included in the field definition of the Axiom query.

Parameter	Description
Filter	Optional. A filter criteria statement to limit the values available for selection. The filter impacts both what displays in the drop-down list, and what is available when searching using the filter box. When using an Axiom query as a source, you can use the filter parameter, or you can define a filter in the Axiom query settings (or both).

Parameters can be listed in any order after the ComboBox tag and the data source name. You do *not* need to indicate omitted parameters with an "empty" semi-colon.

To create the extended ComboBox tag, you can manually type it within a cell, or you can use the Combo Box Tag Helper to assist you in creating a tag. To open the helper dialog, place the basic ComboBox tag within a cell—meaning `[ComboBox;ComboBoxName]`—then double-click the cell.

Note the following when using the extended ComboBox tag:

- The selected value is still placed in the **Selected Value** cell of the component properties. Indirect cell references and form state / shared variable tags can be used as normal.
- In most cases, **Searchable** should be enabled in the component properties so that users can search the list. All columns displayed in the list are included in the search. However, if the list only has a small number of items then this may not be necessary.

  NOTE: Only the first 100 values from the data source are displayed in the drop-down list. If the list contains more than 100 values, then Searchable must be enabled so that users can search for the other values. This limit does not apply when using the regular `[ComboItem]` syntax; the limit only applies to extended tags.

- In the form, the **Initial Text** will show in the combo box until a value is selected. Once a value is selected, the contents of the Selected Value field will show unless a DisplayCell has been specified, in which case the contents of the DisplayCell will show.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- The Select content tag can be used in Formatted Grid components to present drop-down lists within a grid. You may want to do this if your form is primarily grid-based, or if the combo boxes need to be integrated with the other contents of the grid. For more information, see Using drop-down lists in Formatted Grids.
- The ComboBox refresh variable can be used to present drop-down lists in the Web Client filter panel. You may want to do this if the combo box only impacts the data refresh and does not need to be displayed on the form itself with the other form contents.

Date Picker component

The Date Picker component is an interactive component that displays an input box for date selection in the Axiom form. Users can click the calendar icon to select a date from a calendar control. This date selection can be used to impact the Axiom form in some way.

The selected date is returned to the source spreadsheet as an Excel date/time serial number. However, the display of the date in the Date Picker input box is always in an Excel "short date" format, such as 11/4/2013. The exact format will depend on your system locale.

Component properties

You can define the following properties for a Date Picker component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Selected Date	The currently selected date of the date picker. This setting serves two purposes: • It defines the initial value of the component, before a user makes a selection. For example, you may want to use the Excel function TODAY to set the current date as the starting date for the date picker. Alternatively, you can leave this setting blank for no starting date. You can use the **Choose a date** button **[...]** to open a calendar control and select a starting date, or you can modify the Form Control Sheet directly to use a formula or an indirect cell reference (see note below). • When the user selects a date in the Axiom form and it is submitted back to the source file, the user's selection is placed in this cell on the Form Control Sheet (temporarily overwriting any existing value). The selection can then be referenced by other components or in sheet calculations. The selected date is stored in the source spreadsheet as an Excel date/time serial number. Keep in mind that the date picker allows users to clear the date and return a blank value. If the selected date value is being used in calculations and/or to drive other components in the form, make sure to construct the relationship to accommodate a possible blank value. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected date to be read from and written to the specified cell reference instead of directly within the Selected Date cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected date is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).

Item	Description
Earliest Date	Optional. Specify the earliest date that is valid for a user to select in the date picker. If specified, the calendar control will not allow the user to select a date that is earlier than this date. You can use the **Choose a date** button **[...]** to open a calendar control and select a date, or you can modify the Form Control Sheet directly to use a formula. You can also use indirect cell references (as described for the Selected Date); however in this case the component will only read the value from the specified cell, it will not write back any value. **NOTE:** If the earliest date can change using a formula, and the selected date is now invalid due to the changed earliest date, then the date picker will display as blank rather than showing the invalid date.
Latest Date	Optional. Specify the latest date that is valid for a user to enter into the date picker. If specified, the calendar control will not allow the user to select a date that is later than this date. You can use the **Choose a date** button **[...]** to open a calendar control and select a date, or you can modify the Form Control Sheet directly to use a formula. You can also use indirect cell references (as described for the Selected Date); however in this case the component will only read the value from the specified cell, it will not write back any value. **NOTE:** If the latest date can change using a formula, and the selected date is now invalid due to the changed latest date, then the date picker will display as blank rather than showing the invalid date.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Read-Only	Specifies whether the date picker is read-only. • If disabled (default), then the date picker is active, and Axiom form users can select a date. • If enabled, then the date picker is not active, and the selected date displays as read-only. This is intended for situations where you want to dynamically change the date picker from read/write to read-only depending on a certain criteria.
Auto Submit	Specifies whether the Axiom form automatically updates when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user selects a date. If this setting is disabled, then the user must use the Button component in order to update the form for the changed state.

Item	Description
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Date Picker components. Only the generic styles are available. Most date picker styling is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

▶ Interactive behavior

The Date Picker component allows the user to select a date. This date is submitted back to the source file, and written to the **Selected Date** setting on the Form Control Sheet. You might be collecting the date to save to the database, or to impact the state of another component.

If you want the Axiom form to respond to the selected date, then you must set up the file so that another component references the date and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

An Axiom form could contain a chart with data that starts as of a certain date. The date range for the chart could look to the Selected Date field for the Date Picker component to determine the starting date. When the user selects a new date and submits it back to the source file, the date range of the chart changes and then the data changes in response.

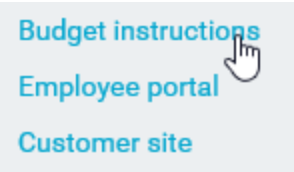
Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- The DatePicker content tag can be used in Formatted Grid components to present a calendar control within a grid. You may want to do this if your form is primarily grid-based, or if the calendar needs to be integrated with the other contents of the grid. For more information, see Using date pickers in Formatted Grids.
- The Calendar refresh variable can be used to present a calendar control in the Web Client filter panel. You may want to do this if the date selection only impacts the data refresh and does not need to be displayed on the form itself with the other form contents.

Hyperlink component

The Hyperlink component can be used to display a hyperlink to one of the following: a web page, an Axiom form, or an Axiom file. Users can launch the hyperlink by clicking the link text on the Axiom form.



Example Hyperlink components in a form

Component properties

You can define the following properties for a Hyperlink component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the hyperlink.
URL	The URL to launch when the button is clicked. The URL must use full HTTP syntax—meaning, use `HTTP://www.axiomepm.com`, not `www.axiomepm.com`. The URL can be to a web page, an Axiom form, or an Axiom file. See Generating a URL to an Axiom form or an Axiom file for more details. The standard HTML "mailto" syntax can also be used here, to open an email with the specified parameters. For example: `mailto:someone@example.com?subject=This%20is%20the%20subject`
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Use New Window	Specifies whether the link is opened in a new window. By default this is enabled, which means the link is opened in a new window. Disable this option if you want the link to open within the same window (replacing the current Axiom form). **NOTE:** Disable this option if you have linked to an Axiom file (to be opened in the Excel Client or the Windows Client) but this Axiom form will be viewed in the Web Client. Otherwise, a new blank window will be opened in the browser in addition to opening the specified file.
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

By default, Hyperlink components display in blue text with no underline. The following styles can be used to change this presentation:

- **text-color**: Removes the blue color and instead displays in the default text color.
- **underlined**: Applies an underline to the text.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Generating a URL to an Axiom form or an Axiom file

If you want to hyperlink to an Axiom form or an Axiom file, you need to be able to provide a URL to that document. The following options are available to obtain a URL:

Option	Works for	Description
Browse to File	Axiom file Axiom form	You can click the **[...]** button to the right of the **URL** box to open the Shortcut Properties dialog. You can then click the **[...]** button to the right of the **Shortcut Target** box to select a file in the Axiom file system. If the selected file is form-enabled, you can choose to open it as an Axiom form by selecting **View as Form** in the shortcut parameters. The shortcut syntax will be converted to a URL when the file is viewed as an Axiom form.
GetDocumentHyperlink	Axiom file only	This function can be used to generate a URL to an Axiom file. You can either copy and paste the URL into the component properties, or you can edit the Form Control Sheet directly to place the function in the URL cell (or reference another cell that contains the function). Note that there is no way to open the file as a form using this method.

Option	Works for	Description
GetFormDocumentURL	Axiom form only	This function can be used to generate a URL to an Axiom form. You can either copy and paste the URL into the component properties, or you can edit the Form Control Sheet directly to place the function in the URL cell (or reference another cell that contains the function). The target file must be form-enabled in order for the URL to be valid. When using this approach, you can use various function parameters to apply a sheet filter, pass variable values, or generate a PDF.
Copy shortcut to clipboard	Axiom file only	You can right-click a file in Axiom Explorer or the Explorer task pane, and then select this option to copy the URL to the clipboard. You can then paste the URL into the component properties. Note that there is no way to open the file as a form using this method.

Remember, if you link to a regular Axiom file, that file will be opened in the Excel Client or the Windows Client. If the Desktop Client is not already installed on the user's computer, Axiom Software will attempt to install it. Therefore you should only link to Axiom files if either of the following is true:

- Users will view the Axiom form within the Excel Client or Windows Client.
- Users will use the Web Client on a Windows PC, and either the Excel Client or Windows Client is already installed on that machine or it is acceptable for the user to install it.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- **Formatted Grids**: You can display hyperlinks within Formatted Grid components, using the HREF content tag. This is especially useful when you want to generate links using an Axiom query, and display the links embedded within the other grid content (such as one hyperlink per row of the grid). See Using hyperlinks in Formatted Grids.
- **Images**: You can link to a URL using an Image component. The user can click on the image to launch the URL.
- **Shapes**: You can link to a URL using the Rectangle or Ellipse components. The user can click on the shape to launch the URL.

Although it is possible to configure a Button component to open a designated URL or a file, this is not recommended. The Button component is not designed to serve as a hyperlink and does not support the full set of hyperlink options.

Image component

The Image component displays an image on the Axiom form, such as a company logo. The image must be in PNG or JPG format, and it must be stored in the Reports Library.

NOTE: If you want to display a background image on the entire form, then you should use the **Background Image** property at the form level instead of placing an image on the canvas and sizing it to fit the entire area. For more information, see Setting the background color or image for an Axiom form.

Component properties

You can define the following properties for an Image component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Image Path	Specifies the image to display in the component. Click the **[...]** button to browse to the image within the Reports Library. If the image is not already saved in the Reports Library, you can right-click a folder and select **Import** to import the image (if you have the appropriate rights to do so). The image must be in PNG or JPG format. **NOTES:** • Users must have permission to the image file in order to see it rendered in the form. It is recommended to create a dedicated Images folder in the Reports Library and store all images in this location. You can grant access to this folder using the Everyone role, or you can create subfolders and grant access to users and roles as needed. • The next time you open this file after saving, the path to the image will be automatically converted into a system-managed document shortcut (you can tell the difference by the presence of a `_tid` parameter on the end of the shortcut). This is to make the file reference "repairable" in cases where the file is renamed or moved. Note that if the path is a result of a formula instead of directly within the cell, then the conversion will not occur and the file reference will not be repairable.

Item	Description
URL	Optional. The URL to launch when a user clicks on the component. The URL must use full HTTP syntax—meaning, use `HTTP://www.axiomepm.com`, not `www.axiomepm.com`. The URL can be to a web page, an Axiom form, or an Axiom file. See Generating a URL to an Axiom form or an Axiom file for more details.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip. If the image path is invalid, the tooltip text displays instead of nothing.
Use new window	If a URL is defined, specifies whether the link is opened in a new window. By default this is enabled, which means the link is opened in a new window. Disable this option if you want the link to open within the same window (replacing the current Axiom form).

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Image components. Only the generic styles are available. Image components do not have component-specific formatting properties, so the only way to apply formatting is to use styles.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Label component

The Label component displays text on the Axiom form, such as for a title or brief explanatory text. Labels can also be used to display colored boxes (with no text).

Component properties

You can define the following properties for a Label component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	Optional. The display text for the label. You can leave this blank if you want to use the label as a design element only (such as a colored and/or bordered box). **NOTE:** This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. If a cell reference is used, then the label will display the text defined in the referenced cell. You must edit the Form Control Sheet directly to enter the cell reference.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Symbol	Optional. The symbol to display on the label in addition to the label text (or instead of text). To select a symbol, click the **[...]** button to open the **Choose Symbol** dialog. Within this dialog, you can scroll through the available symbols, or you can use the filter box at the top to find symbols based on symbol names. For example, you can type `file` to see all of the symbols that have the word "file" in the name. When you have found the symbol that you want to use, select it and then click **OK**. The selected symbol shows in the Form Designer / Form Assistant, and the actual symbol name is written to the corresponding field in the Form Control sheet.
Symbol Position	The position of the symbol relative to the label text (**Left** or **Right**). This setting only applies if a symbol is specified for the label. By default, the symbol displays to the left of the label text.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Label components have many different available styles, such as to display normal text or title text, as well as add-on styles to apply specific font sizes, colors, and text alignment. The color codes used in the font color add-on styles are the same as those used by the grid row and column styles.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Label alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- Label and Rectangle components can both be used to draw a box, but rectangle components do not support text. Both components support the same color and border properties, but Rectangle components support additional options such as dashed borders, rounded corners, and the ability to launch a URL.
- When using a Formatted Grid component, label text for grid data can be placed directly in grid cells.

Panel component

Using the Panel component, you can group multiple components within a single container. This is a design element that can be used to:

- Move and position components as a group within the Form Designer
- Display components within the panel area when the file is rendered as a form

Once you have placed a Panel component on the canvas, you must assign one or more "child" components to that panel. For more information about working with panels, see Using panels to group and position components.

Component properties

You can define the following properties for a Panel component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Title Text	The title text for the component. This text displays in the header bar for the component within the Axiom form, if the title bar is enabled. If the title bar is disabled, then this text does not display at all in the form.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Child Layout	Specifies the layout behavior for child components within the panel: • **Positioned** (default): Child components have defined positions in the same way as regular components, except that child component positions are relative to the panel instead of the overall page. For more information, see Using panels to group and position components. • **Flow**: Child components will automatically flow within the panel from left to right (then down as needed), based on a defined flow order. This behavior is intended for forms such as home pages or dashboards, where the specific position of individual controls or charts may not matter, you just want them to fit on the page in a specified order. For more information, see Auto-flow components in a panel.
Flow Layout Direction	Specifies the direction of the flow layout. Only applies when **Child Layout** is set to **Flow**. • **Left to right** (default): Child components start in the top left corner of the panel and flow right. • **Right to left**: Child components start in the top right corner of the panel and flow left.

Item	Description
Child Padding X Child Padding Y	Defines the x-padding and y-padding between child components when using flow layout behavior. These settings only apply when **Child Layout** is set to **Flow**. • Child Padding X defines the horizontal padding between components. It is applied to the right side of each component. • Child Padding Y defines the vertical padding between components. It is applied to the bottom of each component. The padding can be set in pixels (default) or in percentages. For more information, see Setting the padding between child components.
Overflow	Specifies the behavior if child components extend beyond the panel boundaries. Select one of the following: • **Visible** (default): Child components are visible beyond the panel boundaries. This may cause child components to interfere with the expected display of other components that are not part of the panel (for example, to overlap another component). • **Hidden**: Any part of a child component that extends beyond the panel boundaries will be hidden. This may cause child components to appear "cut off." • **Scroll**: Scroll bars are always present on the panel, regardless of whether they are needed. If child components extend beyond the panel boundaries, the scroll bars are active. • **Auto**: Scroll bars are added to the panel only if child components extend beyond the panel boundaries.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Panel components have access to several useful styles to control the size and position of the panel, as well as properties such as background color.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Radio Button component

The Radio Button component allows the user to select one option from a group of options. The Axiom form can change in some way based on the user's selection.

Radio Button components are only valid in the context of a button group. You must have at least two Radio Button components on your form that belong to the same button group. Only one radio button in a group can be selected at any one time.

Component properties

You can define the following properties for a Radio Button component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the button. Remember that the button text should make sense in the context of the other buttons in the group. It should be clear that the selections are mutually exclusive (only one button in the group can be selected at a time) and what each button will do if selected. You may also want to use a separate Label component to provide an overall title and/or explanatory text for the button group.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.
Button Group	The button group for the component. Radio buttons must belong to a button group. You can define a new button group name by typing the name into the box, or you can select from any previously defined group name. When the Axiom form is rendered, users can select one of the buttons within a particular button group at any one time.

Item	Description
Is Selected	The current state of the button in the button group, selected or not selected. This setting serves two purposes: • It specifies which button is selected within the button group initially, when the user first opens the Axiom form. By default, this is disabled, which means the button is not selected. If you want this button to be selected initially, enable this setting. • When a user views the Axiom form and selects the button (or selects another button in the group, thereby causing this button to become not selected), the change in the selected status will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the current status of this button. Only one button within a button group can be selected at any one time.
Auto Submit	Specifies whether the Axiom form is automatically updated when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user selects a radio button. If this setting is disabled, then the user must use a Button component in order to update the form for the changed state. For example, you might disable the auto-submit behavior if the radio button is one of several user selections that are intended to be submitted together at one time, instead of piecemeal as each one changes. In that situation the user can make all necessary changes for all related components, and then click a Button component to submit the changes at once and trigger an update.
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form.

Item	Description
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Radio Button components. Only the generic styles are available. Most radio button styling is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Radio Button component allows the user to select an option from among all of the radio buttons in the button group. The current state of the radio buttons is submitted back to the source file, and written to the **Is Selected** setting on the Form Control Sheet (for each button in the button group). Only one of the radio buttons in the button group can be selected at any one time.

If you want the Axiom form to respond to the state of the radio buttons, then you must set up the file so that another component references the radio button state and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

An Axiom form could contain a group of radio buttons that determine what layer is currently visible in the form. When a user selects one of the radio buttons, that selection is written back to the source file to the Is Selected setting for all of the radio buttons in the group. The selected radio button would be set to On, and the other radio buttons would be set to Off.

The layer visibility settings could look to the radio button settings to determine when a particular layer would be visible or not. For example, the visibility setting for layer 1 could contain a formula such as:

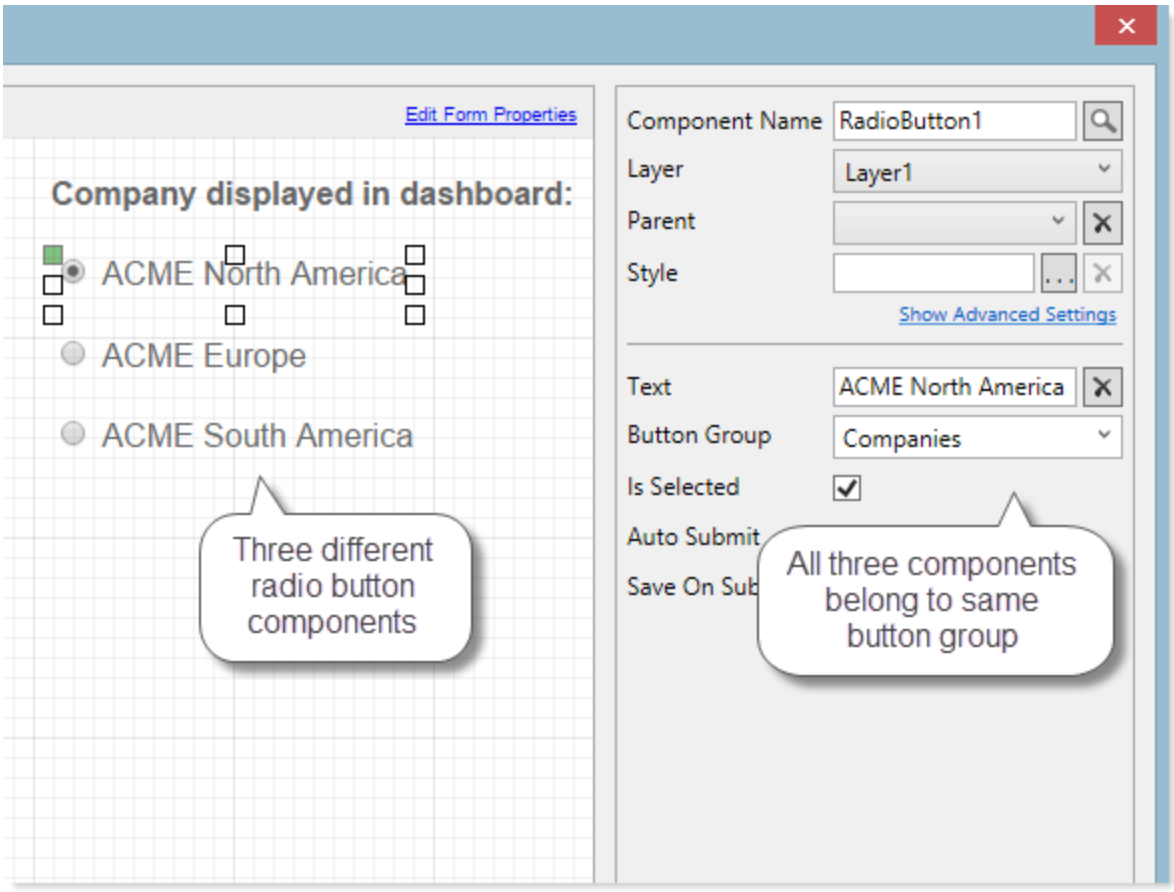
```
=IF(Control_Form!D428="On","On","Off")
```

In this example, the Is Selected setting for the radio button is located on the Form Control Sheet in cell D428. If that radio button is selected, this layer is visible. If that radio button is not selected, this layer is hidden.

Button groups

Radio Button components can only be used as a group of buttons. For example, you might set up an Axiom form with three different radio buttons, asking the user to select which company they want to view in the form.

To do this, you would drag and drop three different Radio Button components onto the form canvas, and then in the component properties you would assign each to the same button group. It is also a good idea to add a Label component, for a title and/or explanatory text.



While setting up the radio buttons on the canvas, you may find it helpful to use **Arrange > Align** to align all of the buttons to one edge (in this example, the left edge), and then to distribute them equally (in this example, vertically).

The radio buttons would display as follows in the Axiom form. In this case, the ACME North America button is selected by default (due to enabling Is Selected in Group). The user can click on another radio button in this group to select that option and clear the ACME North America option.

What occurs when a user selects a radio button is entirely up to the form designer. In this example, each company could be set up on a different layer of the form. The layer visibility settings could look to the radio button settings to determine when a particular layer would be visible or not. For example, the visibility setting for layer 1 could contain a formula that says the layer is visible when Is Selected is On for the ACME North America radio button, and not visible when the setting is Off.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- Button components can also be used in a group as selectors, as an alternative to Radio Button components. For more information and a discussion of the differences between the two options, see Using buttons as selectors (button groups).
- If you want to present more than 3-4 options, a Menu component may be more user-friendly than a button group. Users can select items from the menu in order to impact the form in some way. For more information, see Menu component .

Slider component

The Slider component allows users to slide a button along a designated range of values, which can be used as an interactive component to change something in the Axiom form.

Component properties

You can define the following properties for a Slider component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Slider Value	The initial value for the slider. If you do not specify a value, then the slider will display with the button at the bottom (lowest value) of the scale. By default, the initial slider value is 50, with the default scale set at 1 to 100 (the min and max values). You should edit this setting as appropriate for your defined slider scale. When a user slides the button along the scale and then stops at a particular value, that value will be submitted back to the source file and written into this field.

Item	Description
Orientation	Specifies the orientation of the slider. Select either **Horizontal** (button slides side to side) or **Vertical** (button slides up and down).
Min Value	The minimum value for the slider scale. By default this is 1. You should edit this to be the lowest value that you want users to be able to select in the slider.
Max Value	The maximum value for the slider scale. By default this is 100. You should edit this to be the highest value that you want users to be able to select in the slider.
Step Frequency	Defines the selectable intervals within the slider range. By default this is 2. This option specifies the amount the slider will increase or decrease as the button is moved. For example, if the range is 1 to 10 and the step frequency is 1, this means that moving the mouse will go from 1 to 2 to 3, etc. If the range is 1 to 100 and the step frequency is 5, this means that moving the mouse will go from 1 to 5 to 10 to 15, etc. This setting also determines the selectable values within the range. For example, if the range is 1 to 100 and the step frequency is 5, there is no way for a user to select 37, they can only select 35 or 40.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip. The user must hover over the body of the slider or the slider button in order to see the tooltip. The up and down buttons have built-in tooltips, and if tick marks are shown then hovering over a tick mark shows the corresponding value.
Show Ticks	Specifies whether selectable intervals are shown as tick marks along the slider. By default, this is not selected.
Auto Submit	Specifies whether the Axiom form automatically updates when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user moves the slider button. If this setting is disabled, then the user must use a separate Button component in order to update the form for the changed state.

Item	Description
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Slider components. Only the generic styles are available. Most slider styling is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Slider component allows the user to slide a button along a range of values, and then submits the currently selected value back to the file. The current slider value is written to the **Slider Value** setting on the Form Control Sheet.

If you want the Axiom form to respond to the current value of the slider, then you must set up the file so that another component references the slider value and changes based on it. For more information on

setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

An Axiom form could have a chart or a formatted grid that is configured to show data as of a particular period, and then a slider could be used to allow users to specify which period to use. One of the report sheets in the source file could contain a cell reference to the Slider Value on the Form Control Sheet. The data for the chart or grid could then be set up to change based on the value, for example to return GL2013.YTD3 when 3 is the slider value, GL2013.YTD4 when 4 is the slider value, etc.

A slider could also be used to perform "what-if" analysis. For example, you could have a slider that specifies a particular percentage for a key planning assumption, and then adjust the data in charts and grids based on the selected percentage. The Axiom form could also be configured to save the changed assumption and data back to the database.

Text Box component

The Text Box component displays text on the Axiom form and allows users to edit that text. This text can be used to impact the data in the form in some way, or it can be saved to the database.

Text boxes are always displayed in a bordered box, to signal to the user that the text is an editable field. Text boxes can be used for regular free-form text input, or they can use special features such as:

- Single-line or multi-line input
- Rich text input (font formatting, lists, alignment)
- Numeric-only input within an optional range
- Masked input text to restrict the input to valid characters and to define an input format (such as for a phone number)

Generally speaking, text boxes should only be used when text needs to be edited. Although it is possible to configure a text box as read-only, this option is intended to support dynamically enabling or disabling a text box for editing based on some criteria. Text boxes are not intended for display text that never needs to be edited—for that, you should use a Label component.

Component properties

You can define the following properties for a Text Box component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The text for the text box. This setting serves two purposes: • It defines the initial text for the text box, when the user first opens the form. You should only define text here if you want that text to be the default value for the text box. If you want to display instructional text within the text box, such as "Enter name here", then you should use the **Placeholder** field instead. • When a user views the form and edits the text in the text box, this changed text will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the current state of the check box. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the text to be read from and written to the specified cell reference instead of directly within the Text cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the text is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Placeholder	The placeholder text for the text box. This text is displayed when no Text value is defined for the text box. It is typically used to define instructional text for the text box, such as "Enter name here". As soon as a Text value is defined for the text box, the placeholder text is no longer displayed and instead the Text value is displayed. Keep in mind that placeholder text is not the same as defining a default value for Text. Placeholder text only displays when the value of Text is blank. This also applies when using indirect cell references for Text (if the target cell is blank), or when using a form state / shared variable tag for Text (if no value has been set for the form state key or shared variable). If instead you want Text to have a default value, then you should define that value as the Text value instead of using placeholder text. **NOTE:** The appearance of placeholder text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the placeholder text displays in a lighter color than the Text value, but not always.

Item	Description
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip. **NOTE:** For numeric text boxes, the custom tooltip displays unless validation detects that the current entry is out of bounds in regard to the defined Min or Max. In that case, the validation tooltip displays.
Read-Only	Specifies whether the text box is read-only. • If disabled (default), then the text box is editable, and Axiom form users can change the text in the text box. • If enabled, then the text box is not editable and the text displays as read-only. This is intended for situations where you want to dynamically change the text box from read/write to read-only depending on a certain criteria. If you want text that will always be static, use a Label component instead.
Type	Specify a type for the text box: • **Text** (default): The text box accepts free-form text inputs. When using this type, you can also optionally enable multi-line input or enable rich text input. • **Input Mask**: The text box uses a defined input mask format to restrict the user input to certain character types and formats. Selecting this type exposes additional options that only apply when using the Input Mask type. For more information, see Using an input mask with text boxes. • **Numeric**: The text box only accepts numeric characters within an optional range. Users cannot input letters or other special characters. Selecting this type exposes additional options that only apply when using the Numeric type. For more information, see Using numeric text boxes.
Rich Text	Specifies whether the text box supports rich text. Only applies if **Type** is set to **Text**. By default, this is disabled. This means that the text box is a regular text box, and any text entered into the text box is plain text. For more information on enabling this option and using rich text, see Using rich text boxes.

Item	Description
Multi-Line	Specifies whether the text box allows multiple lines. Only applies if **Type** is set to **Text**, and if the **Rich Text** option is *not* enabled. • If disabled (default), only one line is allowed in the text box. The text does not wrap, and the Enter key cannot be used to create new lines. Instead, the Enter key exits the component and triggers an auto-submit (if Auto Submit is enabled). • If enabled, then the text box allows multiple lines. The text will wrap if it exceeds the width of the component, and the Enter key can be used to create new lines. To exit the component and trigger an auto-submit, you must use the Tab key or click outside of the component.
Auto Submit	Specifies whether the Axiom form automatically updates when a user changes the state of the component. • If disabled (default), then a form update is not triggered when the user edits the text box. This means that you must use a separate Button component (or a different component configured to auto-submit) if you want users to be able to update the form after entering text into the text box. • If enabled, then the Axiom form automatically updates when the user inputs text into the box and then exits the component—such as by clicking the Enter key, the Tab key, or by clicking outside of the component.
Save On Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.

Item	Description
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

NOTE: Styles designed for use with text boxes, such as the **required** style, will not affect the rich text box.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Text Box component allows the user to type text into the box. This text is submitted back to the source file, and written to the **Text** field on the Form Control Sheet.

This text can be used to impact the form in some way (for example, so that the user can enter a free-form filter), or it can be saved to the database.

If you want the Axiom form to respond to the submitted text, then you must set up the file so that another component references the text and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

If you want the submitted text to be saved to the database, then you must set up the file to enable save-to-database. For more information, see Saving data from an Axiom form.

Example

An Axiom form could contain a Text Box component to allow a user to enter their name as part of a larger input form. When all inputs are complete, they will be saved to the database. This is typically accomplished via a separate Button component that can trigger the save-to-database once all inputs are complete.

If the text input is required, then you should enable auto-submit for the text box so that the button can be dynamically enabled once all required inputs are complete. If the input is not required, then you might leave auto-submit disabled unless another component is dependent on the input. It would be rare to enable both auto-submit and save-on-submit for the text box, unless the text input is the only thing to be saved to the database and you want it to be saved immediately.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- Formatted Grids support two ways to allow users to edit text in a grid—simple unlocked cells and the TextArea content tag. The TextArea content tag supports special features similar to the Text Box component, such as placeholder text and multi-line inputs. For more information, see Using text boxes in Formatted Grids.
- The refresh variables String, Integer, and Decimal can be used to present text boxes in the Web Client filter panel, to allow input of text or numeric values. You may want to do this if the user input only impacts the data refresh and does not need to be displayed on the form itself with the other form contents. For more information, see Defining refresh variables for the Web Client Filters panel.

Using rich text boxes

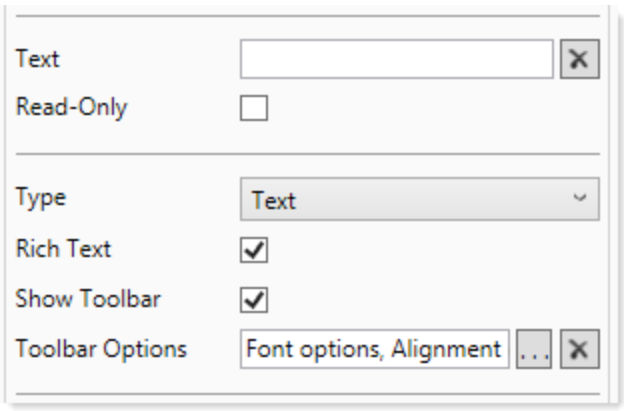
You can configure a Text Box component to enable rich text input. This allows the user to apply a limited set of formatting options to the text, such as bold and italics, ordered and unordered lists, and alignment options.

NOTE: When rich text is enabled, the text box uses a different underlying control than the regular text box. These controls behave and display differently. Additionally, styles designed for the regular text box will not affect the rich text box, and vice versa.

Configuring a Text Box component for rich text

To configure a Text Box component to use rich text, set the **Type** property to **Text** (this is the default setting). You can then configure the following options:

Item	Description
Rich Text	Select this option to enable rich text input for the text box. When users input text, they can optionally apply formatting such as bold and italics. This can be done by using the rich text toolbar, or by using standard shortcut keys (such as CTRL+B for bold).
Show Toolbar	Specifies whether the formatting toolbar displays on the rich text box. This option only applies when the **Rich Text** option is enabled. • If disabled (default), the text box displays as a bordered box, with no toolbar. In this case, users must use shortcut keys to format the text (such as CTRL+B for bold). Other formatting options, such as lists and text alignment, are not available. • If enabled, the text box displays in a framed box, with a toolbar at the top to apply text formatting. You can configure which formatting options display on the toolbar.
Toolbar Options	Specifies which formatting options display on the toolbar, when the toolbar is enabled. The currently selected options display in the Toolbar Options field. To configure the options, click the **[...]** button and then select the check boxes for the desired options: • **Font options**: Enables toolbar buttons for bold, italics, and underline. This option is selected by default. If you disable this option, users can still apply font formatting by using shortcut keys (such as CTRL+B for bold). • **Alignment options**: Enables toolbar buttons for right, left, center, and justified text alignment. • **List options**: Enables toolbar buttons for numbered lists, bullet lists, and indenting. **NOTE:** On the Form Control Sheet, your selections are converted into a numeric code that tells Axiom Software which options to display on the toolbar. It is recommended to always use the Form Assistant or Form Designer to configure the options, to ensure that a valid code is used.



Example rich text box configuration

When rich text is enabled, some other text box properties no longer apply or have special behavior:

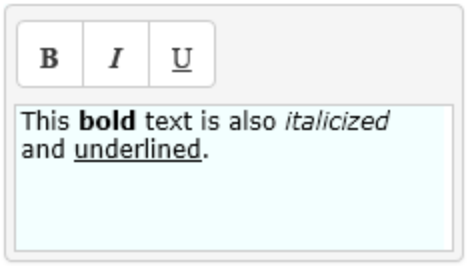
- **Placeholder**: Does not apply when rich text is enabled and will be ignored. Rich text boxes do not support placeholder text.
- **Multi-Line**: Does not apply when rich text is enabled and will be ignored. Rich text boxes always allow multi-line input.
- **Text**: When defining default text for the rich text box, you can use valid HTML formatting tags, such as to apply bold or italics to the text. The easiest way to do this is to let Axiom Software generate the tags for you:
    - Preview the form in a browser and enter the desired text with the desired formatting.
    - Submit the text to the source file (either by enabling auto-submit for the text box, or by using a Button component), and then use the **Download Source Workbook** option on the **Tools** menu.
    - In the downloaded copy, locate the **Text** field for the text box in the Form Control Sheet, and then copy the text that was written to the field with the formatting tags.
    - Go back to the original source file and paste this text into the Text field.

Additionally, all settings relating to the Numeric and Masked Input types do not apply when using the Text type. If defined on the Control Sheet, these options will be ignored. This includes options such as minimum / maximum numeric values and input mask format.

Rich text box behavior

When rich text is enabled for a text box, users can apply formatting by using keyboard shortcuts (such as CTRL+B for bold), or by using the formatting toolbar.

The **Show Toolbar** option determines whether the toolbar displays on the text box. If enabled, then the text box displays as a framed box with a toolbar at the top. For example:



If disabled, then the text box is a simple box with a border. In this case, the user must use keyboard shortcuts to apply font formatting (other formatting options are unavailable without the toolbar).

When the user's text is submitted back to the Text field in the source file, the formatting is indicated using HTML tags. The sentence in the previous example would be written to the Text field as:

```
This <strong>bold</strong> text is also <em>italicized</em> and <span
style="text-decoration:underline;">underlined</span>.
```

Using numeric text boxes

You can configure a text box as numeric, in order to restrict the user input to a number. This input can also be validated against an optional range of valid values. To do this, you must:

- Configure the Text Box component settings to support numeric input
- Set the number format of the Text cell to the desired number format

Configuring a Text Box component for numeric input

To configure a Text Box component for numeric input, set the **Type** property to **Numeric**. You can then configure the following options:

Item	Description
Min	Optional. The minimum number that can be entered into the text box, to communicate the desired numeric range to the user. By default, this option is blank, which means there is no minimum value. Enter a number if you want to define a minimum value. For example, if the minimum is set to 1, then the user can enter any number that is 1 or higher (up to the defined maximum number). If the user enters a number lower than the minimum, such as 0 or -5, then a red validation bar displays on the right side of the text box.
Max	Optional. The maximum number that can be entered into the text box, to communicate the desired numeric range to the user. By default, this option is blank, which means there is no maximum value. Enter a number if you want to define a maximum value. For example, if the maximum is set to 100, then the user can enter any number that is 100 or lower (down to the defined minimum number). If the user enters a number higher than the maximum, such as 150, then a red validation bar displays on the right side of the text box.

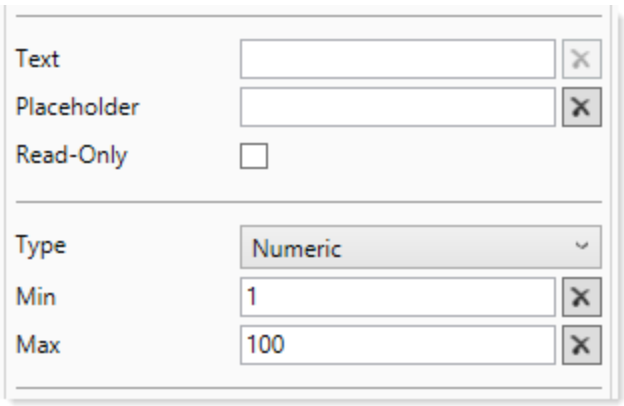
NOTE: Once a minimum or maximum value is set for the text box, the text box cannot be left blank or else it will display with the red validation bar. Therefore, the defined range should not be used with optional values unless there is a default value that you can define for the text box.

You can define just a minimum value or just a maximum value if the range only needs to be validated at one end. For example, the minimum could be set to 0 with no defined maximum, if you only need to verify that users do not enter negative numbers. You can also leave both values blank to allow any number.

If a user enters a number that is out of bounds, the tooltip displays a validation message as follows:

- If both a Min and Max are defined: "The value must be between *Min* and *Max*."
- If only a Min is defined: "The value must be greater than or equal to *Min*."
- If only a Max is defined: "The value must be less than or greater to *Max*."

If you intend the numeric input to be a percentage, the minimum and maximum values can be entered as either decimals (.1) or with percentage signs (10%).



Example numeric text box configuration

IMPORTANT: The minimum and maximum values provide validation messages only; the text box does not prevent the entry of a number that is outside of the range. The user is given feedback about the invalid value, but the value is still submitted back to the source file. There are no built-in controls to prevent the invalid value from being saved to the database or used in any other form processes. If you want to prevent a save-to-database based on the presence of an invalid value, then you must manually build in these controls. For example, you can dynamically enable or disable the button that executes the save-to-database based on whether certain values are valid, or you can use custom save validation to include these controls within the save-to-database process itself.

When the text box is set to Numeric type, all settings relating to the Text and Masked Input types do not apply. If defined on the Form Control Sheet, these options will be ignored. This includes options such as rich text, multi-line input, and input mask format.

Setting the number format for the Text cell

When using the Numeric type for a Text Box component, it is recommended to set the number format of the Text field to one of the following: Number, Currency, or Percentage. To do this, you must format the cell on the Form Control Sheet.

To quickly locate the Text field, you can do the following:

- Select the text box in the Form Designer dialog, or in the canvas thumbnail on the Form Assistant task pane.
- In the component properties, double-click the **Text** label.

This places your cursor on the corresponding Text property in the Form Control Sheet. You can then use regular spreadsheet features to change the number format as desired.

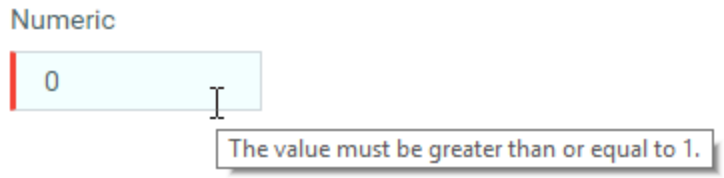
Setting the number format for the cell helps ensure that the numeric input will be handled and displayed as expected. If the cell is set to General or some other non-numeric format, the numeric text box may not behave as expected.

NOTES:

- If you are using an indirect cell reference to read and write the Text value to another cell, then set the number format on that target cell instead of the Text cell. For example, if the Text cell contains `[Input!D10]`, then you should set the number format of cell D10 on the Input sheet to the desired format instead of the Text cell on the Form Control Sheet.
- The number format of the Text field will be honored when the Text field uses a `[SharedVariable]` tag or a `[FormState]` tag, even though the user's input is not actually placed in the cell.
- The number format is honored by the validation message that displays when the user's entry is out of bounds. For example, if the cell is configured as currency and the maximum value is 100, the validation message will show the maximum value using the currency format (such as $100.00).

Numeric text box behavior

Users can type numbers into the text box. If the number does not fall within the defined minimum and maximum range, a red validation bar displays in the text box, and the tooltip displays a validation message.



The validation of the inputted value happens immediately after exiting the text box, regardless of whether the text box is set to auto-submit. It is not necessary to submit the value in order to validate it. As mentioned previously, invalid values are not prevented from submitting and do not prevent any processes such as save-to-database.

If the Text field is set to a numeric cell format (as described in the previous section), then the text box behaves as follows:

- The inputted value displays in the text box according to the defined numeric format. For example, a user may enter `1000` into the cell, which then displays as `$1,000.00` if the cell uses Currency formatting.
- When a user tabs or clicks into the text box in order to edit an existing value, the raw value is displayed and the full contents of the cell are selected. This makes it easy to overwrite the current value with a new inputted value. Users can click in the cell again to de-select the full cell contents and make targeted edits to the value.

- If the cell uses Percentage format, numbers should be entered in decimal format. For example, enter `.1` for `10%`. It is not necessary to enter the percent sign; this will be applied by the cell format.

NOTE: Although users cannot type non-numeric characters into the text box, it is possible to copy and paste any text into the box. Any non-numeric entry will be flagged as invalid.

Using an input mask with text boxes

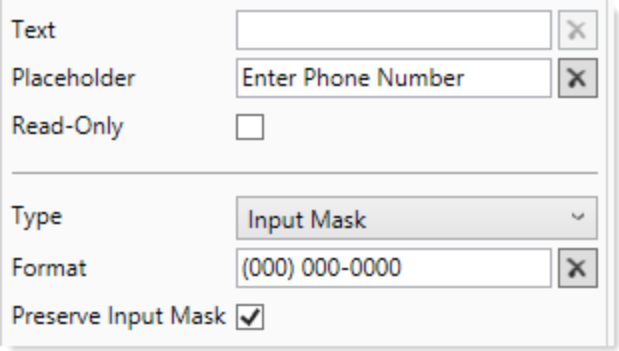
You can configure a text box to use an input mask. An input mask defines the allowed inputs for a text box, and specifies a format for the input.

For example, imagine that you need users to enter a phone number into a text box, to be saved to the database. If you use a regular text box, users might enter phone numbers in a variety of formats, such as 555-555-1111 or 555.555.1111 or (555) 555-1111. Some users might not include the area code, and other users might add an extension. This approach will result in inconsistent data in the database. Instead, you can use an input mask to define the specific allowed input and format for the phone number, so that all entries will be consistent.

Configuring a Text Box component to use an input mask

To configure a Text Box component for numeric input, set the **Type** property to **Input Mask**. You can then configure the following options:

Item	Description
Format	Define the input mask format. For example, you might enter `(000) 000-0000` to specify a phone number format. See the following section for more information on defining an input mask format.
Preserve Input Mask	Specifies whether the text submitted to the source file includes the input mask format or not. • If disabled (default), then the user input will be written back to the source file as raw text with no format. • If enabled, then the user input will be written back to the source file using the specified format. For example, imagine the input mask is defined as (000) 000-000, and the user enters (123) 456-7899. If this option is disabled, then the input will be written to the source file as raw text: 1234567899. If this option is enabled, then the input will be written to the source file as it displays to the user, including the formatting characters. You should enable this option if you need to display the input in a different component using the same format, or if you need to save the input to the database using the format.



Example input mask text box configuration

When the text box is set to Input Mask type, all settings relating to the Text and Numeric types do not apply. If defined on the Form Control Sheet, these options will be ignored. This includes options such as rich text, multi-line input, and minimum / maximum numeric values.

Defining the input mask format

The input mask format defines both the allowed characters for input and the display format for the input. The following tables list the recognized characters that can be used to define an input mask.

Input characters

The following characters determine what a user can enter into the text box:

Character	Description
0	User must enter a number between 0 and 9.
9	User must enter a number between 0 and 9, or a space.
#	Same as 9, except user can also enter plus (+) or minus (-) signs.
L	User must enter a letter from a-z, in upper or lower case.
?	User must enter a letter from a-z, in upper or lower case, or a space.
&	User can enter any character.
C	User can enter any character, or a space.
A	User can enter any alphanumeric character.
a	User can enter any alphanumeric character, or a space.

Display characters

The following characters define display elements of the input mask:

Character	Description
.	Decimal placeholder. The decimal separator will be determined by the current regional format.
,	Thousands placeholder. The display character will be determined by the current regional format.
$	Currency placeholder. The display character will be determined by the current regional format.
All other characters	All other characters not listed as recognized display or input characters will be rendered as is.

The following examples illustrate some potential uses of input mask formats:

`$0,000`

User can enter a 4 digit number. The number will display with the appropriate thousands character and currency character for the regional format.

`00/00/000`

User can enter a date with 2 digits required for the day and month. This enforces entries such as 07 for days and months less than 10. Each segment of the date is separated by a slash.

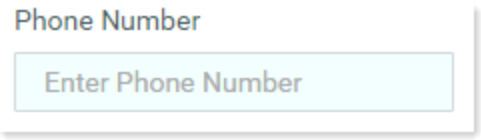
`AAA-AAAAA`

User can enter any combination of numbers or letters, with 3 characters in the first section and 5 characters in the last section, separated by a dash. This could be used for an alphanumeric code in a specific format.

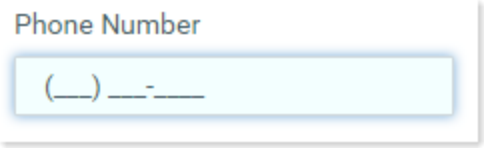
Input mask behavior

The following example of input mask behavior uses an input mask format of (000) 000-000.

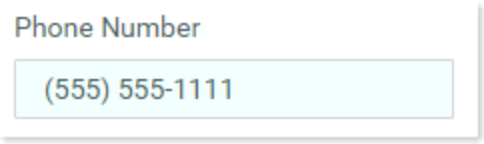
If no text has been input (and no default text is present in the Text field), then the text box displays placeholder text as normal (or blank if no placeholder text is defined).



When the user tabs or clicks into the text box, the input mask displays. The underscores indicate areas where text input is expected.



As the user types in text, the underscores are replaced by the inputted text.



The inputted text always displays in the text box using the defined format mask. The value submitted back to the source file is either formatted text (exactly as shown in the text box) or raw text (only the user input with no formatting), depending on whether **Preserve Input Mask** is enabled.

NOTES:

- If Preserve Input Mask is enabled, then inputted characters remain at the location they are input, even if missing characters are present before the inputted character. If Preserve Input Mask is not enabled, then any missing characters are not honored and the inputted characters will be moved over to fill the empty space.
- It is not currently possible to require the user to fill in all of the designated input characters. In the previous example, the user could fill in 555 and stop, which would result in an incomplete phone number. However, this issue would be visually obvious to the user, as the missing inputs would continue to display in the text box with underscores.

Toggle Switch component

The Toggle Switch component is an interactive component that displays an on/off switch on the Axiom form. Users can click the switch to toggle it on or off, to enable or disable something in the form, or to otherwise toggle between two states.

Example toggle switches

NOTE: The Web Client Container must be enabled for the form in order to use this component. If the Web Client Container is not enabled, an error message will display on the component in the Form Designer, and when the form is rendered the component will display as a check box instead of a toggle switch.

Component properties

You can define the following properties for a Toggle Switch component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Is Checked	The current state of the toggle switch, checked (On) or not checked (Off). This setting serves two purposes: • It specifies the initial state of the toggle switch, when the user first opens the form. By default, this is disabled, which means the toggle switch is set to Off. If you want the toggle switch to be set to On initially, then enable this setting. • When a user views the form and toggles the switch on or off, this state change will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the current state of the toggle switch. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the checked status to be read from and written to the specified cell reference instead of directly within the Is Checked cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the checked status is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
On Text	Optional. Defines text to display when the switch is toggled to On. By default, the text **On** is used if no alternate text is defined.
Off Text	Optional. Defines text to display when the switch is toggled to Off. By default, the text **Off** is used if no alternate text is defined.
Tooltip	Optional. The tooltip text for the component. When a user hovers the cursor over the component, the text displays in a tooltip.

Item	Description
Auto Submit	Specifies whether the Axiom form is automatically updated when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user toggles the switch on or off. If this setting is disabled, then the user must use a Button component in order to update the form for the changed state. For example, you might disable the auto-submit behavior if the toggle switch is one of several user selections that are intended to be submitted together at one time, instead of piecemeal as each one changes. In that situation the user can make all necessary changes for all related components, and then click a Button component to submit the changes at once and trigger an update.
Save on Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.
Enabled	Specifies whether the component is enabled. By default this is set to On, which means that the component displays normally and users can interact with it (if applicable). This setting can be used to dynamically enable or disable the component using a formula. If set to Off, then the component displays as grayed out. If the component is normally interactive, users cannot interact with the component while it is disabled. Disabled components cannot trigger update events for the form. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Toggle Switch components. Only the generic styles are available.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Toggle Switch component allows the user to toggle a switch between On or Off. The current state of the toggle switch is submitted back to the source file, and written to the **Is Checked** setting on the Form Control Sheet.

If you want the Axiom form to respond to the state of the toggle switch (on or off), then you must set up the file so that another component references the toggle switch state and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

An Axiom form could contain a column chart with three possible series. By default the chart shows two series, but if the toggle switch is set to On, the third series is shown. The series tag for the third series could be set up using an IF function so that the series only displays if the toggle switch is selected. For example:

```
=IF(Control_Form!D298="Off","No Show","[Series]")
```

In this example, the Is Checked setting for the toggle switch is located on the Form Control Sheet in cell D298. Therefore if the toggle switch state is Off, then this cell will contain the text "No Show," which means the series of data in this row will not display in the chart. But if the toggle switch state is On, then the series tag will display and therefore the associated data will display in the chart.

Design alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- The CheckBox content tag can be used in Formatted Grid components to present a toggle switch within a grid. You may want to do this if your form is primarily grid-based, or if the toggle switches need to be integrated with the other contents of the grid (such as displaying a toggle switch on each row of a grid). There is *not* a separate tag for toggle switches; instead there is a parameter on the CheckBox tag that determines whether it displays as a check box or a toggle switch. For more information, see Using check boxes in Formatted Grids.

Using Grids

Axiom forms support two different types of grids:

- The Formatted Grid component displays data from the source spreadsheet, and supports various formatting and user input features.
- The Data Grid component queries data directly from the database, and displays it in a standardized data grid.

Generally speaking, Data Grid components should be used to display reporting data, and Formatted Grid components should be used to gather data inputs. Formatted Grid components can also be used to display reporting data, in cases where advanced query structures or complex formatting is needed.

In the Form Designer, both components are available in the **Form Controls** section along the left-hand side of the screen.

Formatted Grid component

The Formatted Grid component provides the ability to display information in a formatted grid structure within an Axiom form. The contents to be displayed in the grid are defined by placing a Grid data source in a sheet. You then populate the rows and columns of the data source with the information that you want to display in the grid. In addition to displaying text and numbers, special format tags can be used in the grid in order to display controls such as text boxes, combo boxes, and buttons within the grid cells.

Each Formatted Grid component in a form can look and act very differently depending on the format style used (spreadsheet or thematic), and depending on the features used within the grid.

Formatted Grid components can be used in many different ways to support data inputs and other interactivity. For example, grids can be used as follows:

- As selector tools, to change the form in some way based on the currently selected row in the grid.
- As input grids, with controls such as text boxes, check boxes, and drop-down lists all presented directly within the cells of the grid.
- As navigation tools, to present lists of hyperlinks to other files and forms.

Formatted Grid components also support special features, such as a full symbol library for use in grid cells, the ability to display sparkline charts and bullet charts inline with data, and the ability to edit grid data in a spreadsheet interface.

Data Grid component

The Data Grid component provides the ability to display data from the database in a standardized grid within an Axiom form. The columns to be displayed in the grid are defined by placing a DataGridColumns data source in a sheet. When the grid is rendered in the Axiom form, it queries data from the database based on the component settings and the columns listed in the data source. This direct database query is much more efficient than reading data from the source spreadsheet, and provides better report performance as compared to a Formatted Grid component.

Data Grid components use a standard format that provides a consistent user experience across all forms where it is used. The grid provides built-in tools for paging, filtering, sorting, and drilling data. Data can also be displayed in expandable/collapsible groupings.

Although Data Grid components are primarily for data display, they do support some interactivity. Icons can be displayed in the grid and can optionally be used to trigger a command or open a designated URL. Data grids can also be used as selector tools, to change the form in some way based on the currently selected row in the grid.

Data Grid component

Using the Data Grid component, you can query data from the Axiom Software database and display that data in a grid within an Axiom form. This component is intended to be used to display reporting data.

The Data Grid component queries the data directly from the database, using the primary table defined in the component properties and the columns defined in the associated data source. The resulting data is not returned into the spreadsheet source file; it is only returned into the form. This provides a more efficient and performant method of displaying data in an Axiom form, as compared to the alternate method of querying data into the spreadsheet source file and then tagging it for display in a Formatted Grid component.

The Data Grid component also supports the following reporting features:

- **Drilling:** You can optionally enable drilling for the grid. Users can drill down any row in the grid, to see the data in the row at a different level of detail.
- **Icons and commands:** You can optionally display icons in the grid. The icons can be used simply as informational signals, or they can be used to trigger a command or open a designated URL. The icons can be persistent in the grid, or they can display on hover only. Conditions can be defined for the icons, so that the icons only display when certain conditions are met.
- **Excel export:** You can optionally enable the ability to export the grid contents to an Excel spreadsheet.
- **Built-in grid tools:** When viewing the grid, users can sort and filter the data, move between paged data, and use other built-in tools.

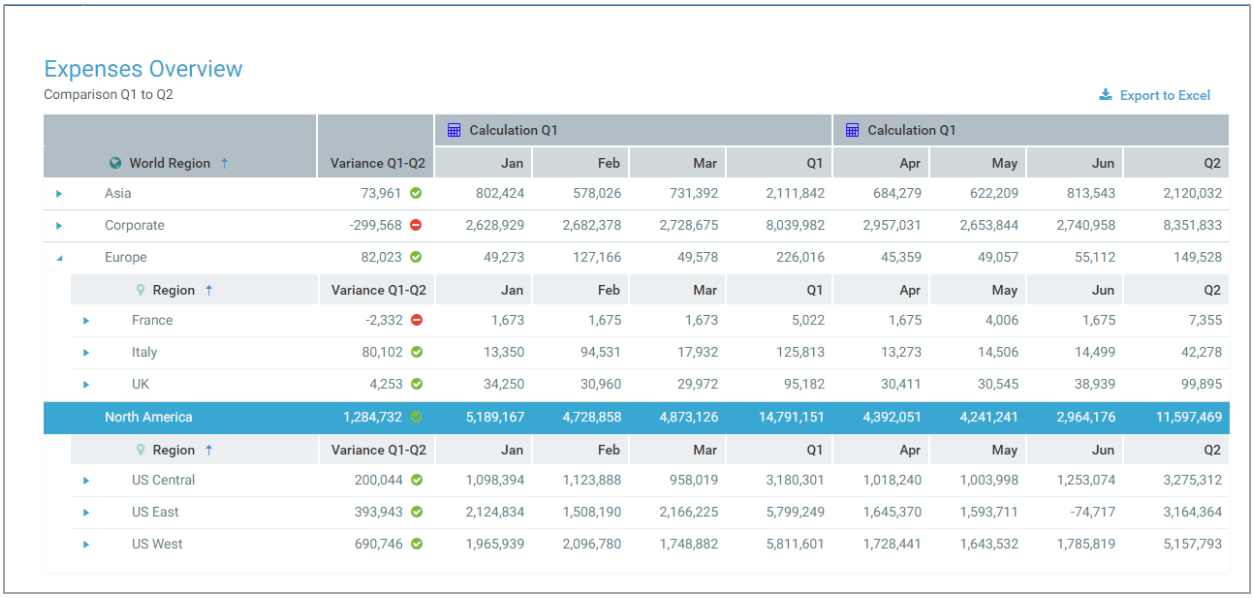
Data Grid components display read-only data. Data grids do not support editing values or displaying other interactive elements such as check boxes or drop-down lists. If data edits are necessary, use a Formatted Grid component instead.

Defining a data grid is a two-part process that requires the following:

- Creation of a DataGridColumns data source in the spreadsheet that defines the columns to display in the grid, as well as other properties such as the sum by level and frozen columns.
- Placement and configuration of a Data Grid component on the Axiom form canvas. The primary table for the query and the overall data filter are defined in the component properties.

Data Grid components also support an alternate data source, HierarchicalGrid, that can be used to show grouped data in the grid.

Generally speaking, the Data Grid component does not support user-definable formatting options. You can define the column width, alignment, and numeric formatting. All other formatting is defined by the grid and cannot be changed.



Expenses Overview

Comparison Q1 to Q2

Export to Excel

World Region	Variance Q1-Q2	Calculation Q1 Jan	Feb	Mar	Q1	Calculation Q1 Apr	May	Jun	Q2
Asia	73,961	802,424	578,026	731,392	2,111,842	684,279	622,209	813,543	2,120,032
Corporate	-299,568	2,628,929	2,682,378	2,728,675	8,039,982	2,957,031	2,653,844	2,740,958	8,351,833
Europe	82,023	49,273	127,166	49,578	226,016	45,359	49,057	55,112	149,528
Region	Variance Q1-Q2	Jan	Feb	Mar	Q1	Apr	May	Jun	Q2
France	-2,332	1,673	1,675	1,673	5,022	1,675	4,006	1,675	7,355
Italy	80,102	13,350	94,531	17,932	125,813	13,273	14,506	14,499	42,278
UK	4,253	34,250	30,960	29,972	95,182	30,411	30,545	38,939	99,895
North America	1,284,732	5,189,167	4,728,858	4,873,126	14,791,151	4,392,051	4,241,241	2,964,176	11,597,469
Region	Variance Q1-Q2	Jan	Feb	Mar	Q1	Apr	May	Jun	Q2
US Central	200,044	1,098,394	1,123,888	958,019	3,180,301	1,018,240	1,003,998	1,253,074	3,275,312
US East	393,943	2,124,834	1,508,190	2,166,225	5,799,249	1,645,370	1,593,711	-74,717	3,164,364
US West	690,746	1,965,939	2,096,780	1,748,882	5,811,601	1,728,441	1,643,532	1,785,819	5,157,793

Example Data Grid component in an Axiom form

NOTE: The Data Grid component has special update behavior that does not follow the same rules as other form components. If you want the data in the grid to change based on user inputs, you must be aware of this behavior and design accordingly. For more information, see Update behavior.

Data source tags

A Data Grid component must have a defined data source within the file to indicate the columns of data to display in the grid. The tags for the data source are as follows:

Primary tag

[DataGridColumns;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Data Grid component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Column]

Each row flagged with this tag specifies a database column to include in the data grid.

[CalculatedColumn]

Each row flagged with this tag defines a custom calculation to include in the data grid. This can be used to display totals, differences, percentages, and other calculations.

[ColumnGroup]

Use this tag to define the start of a column group. This adds another level to the column header area, so that you can define additional header text that spans over all columns in the group. All columns after this tag belong to the group, until an `[EndGroup]` tag is reached. Groups can be nested. For more information, see Creating column groups for the grid header.

[EndGroup]

Use this tag to end a column group. If multiple column groups exist, the group closest to this tag is ended. This tag must have a corresponding `[ColumnGroup]` tag, or else an error occurs. This tag is optional if you want the column group to extend to the end of the column list. For more information, see Creating column groups for the grid header.

Column tags

[ColumnName]

For [Column] rows, specify the fully qualified Table.Column name to include in the grid. The column must be valid to include in the query, based upon the primary table specified in the component properties. The same columns that would be valid to include in the field definition of an Axiom query are valid for inclusion here. You can use regular table columns, calculated fields, and column alias names.

NOTE: If only the column name is specified, the primary table is always assumed as the table, even if the column is validated. For example, if you specify the column as Dept and the primary table is GL2018, then the column used is GL2018.Dept, not Dept.Dept. This is different than the behavior of Axiom queries, which will use the lookup table (Dept.Dept) in this situation. It is recommended to always use fully qualified column names to avoid any confusion.

If you want to display only icons in the column, then you can leave the column name blank. In this case either the [Icon] property or the [HoverActions] property must specify the icons to display.

For [CalculatedColumn] rows, you can enter any unique name. It is recommended to define a name that describes the purpose of the calculation.

[Icon]

Optional. Enter one of the following:

- The name of a single icon to display in each row of the column.
- The name of an IconConfig data source that defines the icons to display in the column, as well as additional icon features.

The valid icon names are the same names allowed for symbols in Formatted Grid components (as well as Label and Button components). You can use any of these features to look up the desired icon name.

> **TIP:** You can right-click the cell and select **Insert Formatted Grid Tag > Symbol**, then use the Tag Editor to select a symbol name (such as `fa-file-o` for a file symbol). You can then copy and paste the symbol name out of the Tag Editor and into the `[Icon]` column.

When listing a single icon name, you can optionally specify a color for the icon, using the syntax *`IconName;ColorName`*. For example: `fa-heart;red`. You can specify the color using a color name (red), a hexadecimal color code (#FF0000), or an Axiom style color code (A32). When using an IconConfig data source name, the color is specified within the data source.

If you want to use additional icon features—such as displaying multiple icons, conditionally displaying icons, or assigning an action to icons—then you must use an IconConfig data source. For more information on creating and using the IconConfig data source, see Using the IconConfig data source.

If you want the column to only contain icons, then the row should be a `[Column]` row and the `[ColumnName]` property should be left blank. In this case, the icons honor the `[Alignment]` property directly to determine the alignment of the icons. If the alignment is set to default, the icons are left-aligned.

If the row has a defined database column name or a calculation, then the icons display along with the column values. In this case the placement of the icons is as follows, depending on the column alignment:

- For left and center alignments, the icons display on the left side of the other column contents.
- For right alignment, the icons display to the right side of the other column contents.

[Header]

The name to display in the grid header for the column. If left blank, the `[ColumnName]` value is used.

For `[ColumnGroup]` rows, this defines the header text to display over the column group. If left blank, a blank header row is displayed over the column group.

[HeaderIcon]

Optional. The name of an icon to display in the column header. You can use the same icon names as for the `[Icon]` column (including appending the optional color).

If no header text is defined, then the icon displays by itself and honors the `[HeaderAlignment]` property directly. If header text is defined, then the icon displays to the left of the header text if the header alignment is left or center, and to the right of the header text if the header alignment is right.

An `[IconConfig]` data source *cannot* be used here; only a single icon name can be used.

[HeaderAlignment]

Optional. The alignment of the header text. Enter any of the following: `Default`, `Left`, `Right`, or `Center`. If left blank, Default is assumed.

By default, the header text uses the same alignment as the column contents (as determined by the `[Alignment]` property). This setting can be used to apply a different alignment to the header text.

For `[ColumnGroup]` rows, the default alignment is Center.

[HoverActions]

Optional. The name of an `[IconConfig]` data source, in order to display icons when the user hovers their cursor over the column contents and perform actions by clicking the hover icons. For more information on creating and using the data source, see Using the IconConfig data source.

Hover icons are designed to display on the opposite side of other column contents. If the column alignment is right, the hover icons display on the left, and vice versa. If the column alignment is center, the hover icons display to the right of the other column contents. (This "opposite" alignment still applies if the icons are the only content in the column.)

[IsVisible]

Determines whether the column is visible in the grid (True/False). You can use this property to dynamically hide and show certain columns, or to include the column in the query but not display it in the grid. If a column is not visible but it is specified as a "sum by" column or as a sort column, then it will still be included in the data query and will impact the results. False is assumed if left blank.

Columns are visible in the grid in the order they are defined in the data source, with frozen columns displayed first, followed by all other unfrozen columns.

If you want to dynamically exclude a column from the data source entirely, then you must use formulas to hide or show the row tag.

[DisplayFormat]

Optional. Defines a display format for the column contents. This is primarily intended to be used when you want to combine the contents of multiple columns together. For example, if you have a column for `Dept.Dept` but you want to display the description in the same column as the department code, you can define a display format as follows:

```
{Dept.Dept} - {Dept.Description}
```

Use fully qualified Table.Column syntax and place column references in curly brackets. The display format can include additional text and characters, such as the hyphen in the previous example. Any column listed in the display format must be valid in the context of the primary table.

If a display format is defined and the column is sorted, it is sorted using the display format instead of the base column values.

[SortOrder]

Specifies whether the grid is sorted by the values in this column. Enter a number that indicates the order of the column in the sort. For example, to sort by a single column, enter 1 for that column. To sort by two columns, enter 1 for the first column to designate it as the primary sort, and 2 for the other column to designate it as the secondary sort. All columns that are not included in the sort order should be left blank. If no columns have an assigned sort order, then the grid is sorted based on the sum by columns.

If a column has an assigned sort order, it is always included in the query, regardless of whether it is visible. This means you can use a column to define the sort but not show that column in the grid.

This property defines the initial sort of the grid. Within the form, users can sort by any column by clicking the column header.

[SortDirection]

Specifies the direction of the sort, `asc` for ascending or `desc` for descending. Ascending is assumed if left blank. Only applies if the column has an assigned number in the `[SortOrder]` column.

[IsSumBy]

Specifies whether the column defines the "sum by" level for the query (True/False). The column must be valid for use as the sum by, based upon the primary table specified in the component properties. The same columns that would be valid to use as the sum by for an Axiom query are valid here. False is assumed if left blank.

For example, if the sum by level is `Dept.Dept`, then each row in the grid represents the sum of data per unique department. If the sum by level is `Dept.Dept` and `Acct.Acct`, then each row in the grid represents the sum of data per unique department / account combination.

If multiple columns are specified, the combined sum by is applied using the order of the columns in the data source. For example, if `Dept.Dept` and `Acct.Acct` are both specified as sum by columns, but Dept is first in the data source, then the sum by is applied as `Dept.Dept, Acct.Acct`.

The data source must contain at least one column that is enabled for use as the sum by. It is not possible to "assume" the sum by level for a data grid. If no columns are enabled, an error will occur when attempting to render the grid.

If **Show Hierarchical Data** is enabled in the component properties, then the sum by columns determine the hierarchical grouping levels for the grid. In this case, there must be at least two sum by columns, and the sum by columns must be in the intended hierarchical order within the data source, with the top-level group listed first. For example, if you are grouping by Country > Region > Dept, then `Dept.Country` must be the first sum by column listed in the data source.

If a column is designated as a sum by column, it is always included in the query, regardless of whether it is visible. Generally speaking, sum by columns should always be visible. If **Show Hierarchical Data** is enabled and a column is designated as a sum by column, that column is automatically visible.

[Width]

Optional. The width of the column in the grid, in pixels. If left blank, the default column width is as follows, depending on the column type:

- Numeric, Date, or Boolean: 120
- Integer (all variations), Identity, or DateTime: 150
- String: 200

All other columns, such as those containing calculations or only icons, default to 200.

[NumericFormat]

Optional. A valid Excel numeric format string to define the number format used by the column. Only applies to columns with numeric data.

To define a display format, enter a valid Excel formatting string. These strings can be obtained as follows:

- Format a cell in a spreadsheet to use the desired display format.
- In the **Format Cells** dialog, on the **Number** tab, select the **Custom** category and copy the string in the **Type** box.

For example, this is the formatting string for a Currency format that shows the negative numbers in parentheses: `$#,##0.000_);($#,##0.000)`

Colors (such as red font for negative numbers) are not supported. Additionally, text replacement strings are only supported for zero values. Other advanced or unusual formats may not display as expected, so be sure to verify the column display.

If you do not define a custom display format, then the default formatting for the column's specified numeric type will be used.

`[CalculatedColumn]` rows use the Number numeric type by default. If you do not want this format, you must enter a format string for the column.

[Alignment]

Optional. The alignment of the column values. Enter any of the following: `Default`, `Left`, `Right`, or `Center`. If left blank, Default is assumed.

The default alignment is as follows:

- Values in frozen columns are left-aligned.
- Values in non-frozen columns are left-aligned for strings and right-aligned for numbers.

[IsFilterable]

Specifies whether users can filter the rendered grid by the aggregated values in the column (True/False). If True, then filtering controls are available on the column header in the grid. These controls are visible when a user hovers over the column header.

[IsFrozen]

Specifies whether the column is frozen at the left-hand side of the screen for scrolling purposes (True/False). If True, then the column displays in the frozen area, before any unfrozen columns, regardless of its placement in the data source. Within the frozen area, frozen columns display in the order they are defined in the data source.

If **Show Hierarchical Data** is enabled in the component properties, then frozen columns do not apply and the `[IsFrozen]` column is ignored. Columns cannot be frozen when using hierarchical groupings in the grid.

[Aggregation]

Optional. Specifies the aggregation type used to aggregate data in the column. In most cases this should be left blank to use the default aggregation for the column—for example, to sum data columns. Aggregation only applies to `[Column]` rows.

If you want to override the default aggregation type for a column, specify one of the following: `Count`, `DistinctCount`, `Min`, `Max`, `Sum`, or `Avg`. The behavior and requirements are the same as when using alternate aggregation with an Axiom query field definition.

[ColumnFilter]

Optional. Specifies a filter to limit the data queried for the column. Enter any valid filter criteria statement. The behavior and requirements are the same as when defining a column filter for an Axiom query field definition. Column filters only apply to `[Column]` rows.

Defining a column filter is different than enabling filtering in the grid using `[IsFilterable]`. The column filter is part of the database query and limits the data returned into the grid for this column only. The filter controls on the grid allow ad hoc filtering on the displayed values in the column.

NOTE: If you want to apply a filter to the entire grid, not just a single column, use the **Data Filter** option in the component properties instead.

[Calculation]

Defines the calculation to use for the calculated column. Only applies to `[CalculatedColumn]` rows.

Enter the desired calculation as a text string, without an equals sign. The calculation must consist of valid database column names and one or more of the following operators: addition (+), subtraction (-), multiplication (*), division (/), remainder (%), or unary negation (-). For example:

`GL2018.M1+GL2018.M2`

This calculation displays the sum of the two columns for each row.

Use parentheses to determine calculation order, such as: `(GL2017.Q1-BGT2017.Q1)/BGT2017.Q1`.

The calculation can use regular table columns, calculated fields, column alias names, and numbers. Table columns and calculated fields must use full Table.Column syntax. You can use any database column that would be valid for inclusion in the data source, though the column does not have to be in the data source in order to be used in the calculation.

[SelectedRowValue]

System-controlled field. This field is populated with the corresponding value for this column, based on the currently selected row within the grid. This field only applies if **Enable Row Selection** is enabled in the component properties. This field is automatically updated by Axiom Software when a user selects a row in the grid.

When a user selects a row in the grid, the value in that row for each column is written to the `[SelectedRowValue]` field. For example, if the user selects the row containing Dept 40000 and Acct 2000, then the value 40000 is written to the data source for the `Dept.Dept` column, and the value 2000 is written to the data source for the `Acct.Acct` column. You can set up the form to use these selected values in some way, such as to show detailed information about the current row. For more information, see Interactive behavior.

NOTE: It is not possible to define a default selected row value for the grid. When the grid is initially rendered, any values in the `[SelectedRowValue]` column are ignored.

[ActionRowValue]

System-controlled field. This field is populated with the corresponding value for this column, based on the row where an action was triggered by clicking on an icon. This field only applies if icon actions are being used in the `[Icon]` or `[HoverAction]` fields. This field is automatically updated by Axiom Software when a user clicks an interactive icon in the grid.

When a user clicks an interactive icon in the grid, the value in that row for each column is written to the `[ActionRowValue]` column. For example, if the user clicks on an icon in the row containing Dept 40000 and Acct 2000, then the value 40000 is written to the data source for the `Dept.Dept` column, and the value 2000 is written to the data source for the `Acct.Acct` column. These values can be referenced by the icon action, such as to display more information about the current department in a Dialog Panel.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

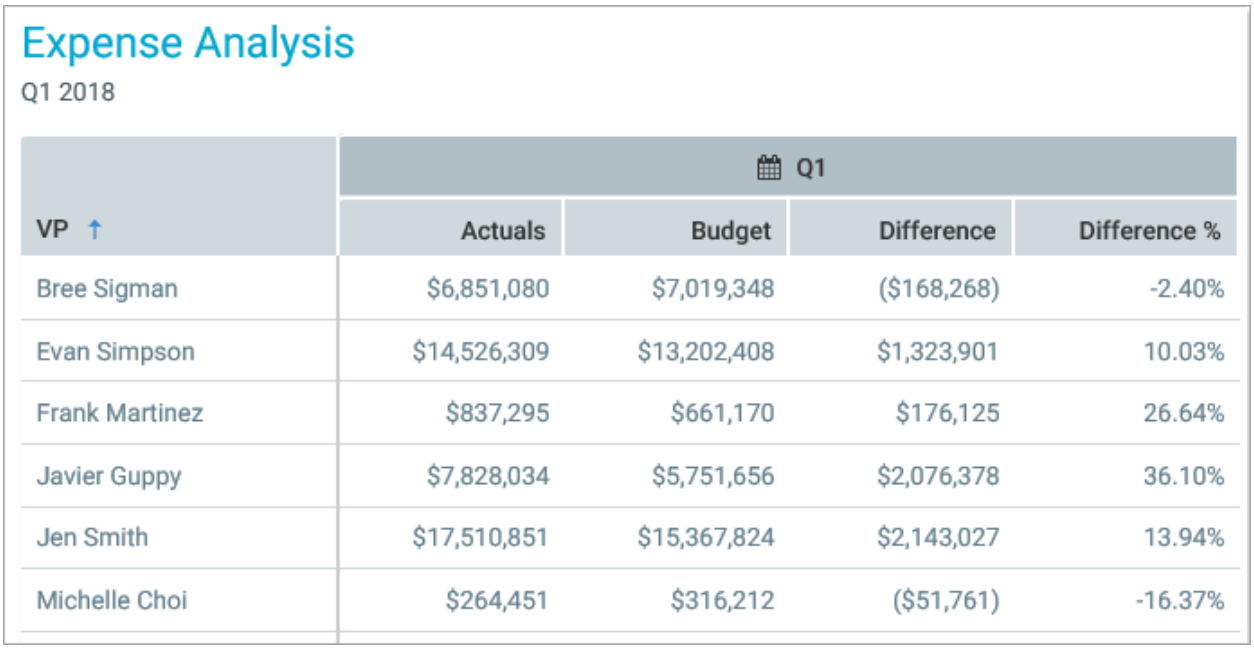
To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Data Grid**. You can also highlight a range of data first and then use the wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The following example shows a sample DataGridColumns data source tagged in a sheet:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
4														
5		[DataGridColumns;Q1Revenue]	[ColumnName]	[Header]	[HeaderIcon]	[IsVisible]	[SortOrder]	[IsSumBy]	[Width]	[NumericFormat]	[IsFilterable]	[IsFrozen]	[Calculation]	[HeaderAlignment [A
6		[Column]	Dept.VP	VP		TRUE	1	TRUE	175			TRUE		
7		[ColumnGroup]		Q1	fa-calendar									
8		[Column]	GL2018.Q1	Actuals		TRUE			125	$#,##0_);($#,##0)				
9		[Column]	BGT2018.Q1	Budget		TRUE			125	$#,##0_);($#,##0)				
10		[CalculatedColumn]	Difference	Difference		TRUE			125	$#,##0_);($#,##0)	TRUE		GL2018.Q1-BGT2018.Q1	
11		[CalculatedColumn]	PDifference	Difference %		TRUE			125	0.00%	TRUE		(GL2018.Q1-BGT2018.Q1)/BGT2018.Q1	

Example DataGridColumns data source

The resulting data grid for the example data source shown above looks as follows:



Expense Analysis

Q1 2018

VP	Q1			
	Actuals	Budget	Difference	Difference %
Bree Sigman	$6,851,080	$7,019,348	($168,268)	-2.40%
Evan Simpson	$14,526,309	$13,202,408	$1,323,901	10.03%
Frank Martinez	$837,295	$661,170	$176,125	26.64%
Javier Guppy	$7,828,034	$5,751,656	$2,076,378	36.10%
Jen Smith	$17,510,851	$15,367,824	$2,143,027	13.94%
Michelle Choi	$264,451	$316,212	($51,761)	-16.37%

Example data grid

Component properties

You can define the following properties for a Formatted Grid component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the grid. You must have defined the data source within the file using the appropriate tags in order to select it for the grid. You can select either of the following types of data sources: • A DataGridColumns data source. This is the basic grid setup that can handle either grouped or flat data. All component properties are available for configuration. • A HierarchicalGrid data source. This is the advanced setup for grouped data. A limited set of component properties are available for configuration (the rest are defined within the data source instead of on the component). For more information, see Using the HierarchicalGrid data source. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Title Text	The title text for the component. This text displays in the title bar for the component within the Axiom form, if the title bar is enabled.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.

Item	Description
Primary Table	The primary table for the data query. Enter any valid table name from the Table Library. The primary table determines which table columns are valid to include in the data grid. System tables (such as Axiom.Columns) cannot be used as the primary table. For example, if you specify GL2018 as the primary table, then the query can retrieve data from that table, plus any reference tables that the primary table looks up to. If you want to include data from multiple data tables, you can include any table that shares keys with the primary table, as well as any shared lookup reference tables. If the component uses a HierarchicalGrid data source, this option does not display. Instead, each grouping level has its own primary table defined within the data source.
Data Filter	Optional. A filter to limit the data returned by the query and displayed in the grid. Enter a filter criteria statement that is valid in the context of the primary table. If no filter is defined, all data that matches the query is displayed in the grid. If the component uses a HierarchicalGrid data source, this option does not display. Instead, each grouping level has its own data filter defined within the data source.
Page Size	Determines how many rows are shown in each page of the grid. By default, the page size is 50. If the results returned by the query exceed the page size, then the grid data is separated into multiple pages. Users can use the page controls at the bottom of the grid to move among pages. If set to 0, all rows display on the same page. Blank is interpreted as the default page size. If the component uses a HierarchicalGrid data source, this option does not display. Instead, each grouping level has its own page size defined within the data source.

Item	Description
Show Hierarchical Column Data	Specifies whether data in the grid is grouped based on hierarchical dimensions. • If enabled, the grid is grouped based on the sum by columns for the grid. The first sum by column determines the top-level grouping, the next sum by column determines the next level grouping, and so on. At least two sum by columns must be specified when grouping is enabled. • If disabled, all data returned by the query is displayed in a flat list with no grouping. For more information, see Showing data grouped by dimensions. If the component uses a HierarchicalGrid data source, this option does not display and does not apply.
Enable Excel Export	Specifies whether users can export the grid contents to an Excel spreadsheet (XLSX). • If enabled, an **Export to Excel** button displays over the top right corner of the grid, so that users can export the grid contents. • If disabled, the button does not display. When a user clicks the **Export to Excel** button, the contents of the grid are exported to an Excel spreadsheet. Configured number formats are not preserved, but default number formatting is applied based on the column data type. User changes to the grid, such as changing the sort order or filtering a column, are not preserved. However, if a refresh variable is used to filter data in the grid, this is preserved. The name of the exported file is the **Title Text** for the component, if defined. Otherwise, a system generated name is used. It is recommended to define title text for this purpose when using the export feature, even if the title bar is not enabled. The following features are not supported with the export feature: • **Hierarchical groupings:** When using **Show Hierarchical Column Data**, groupings are disabled and data is exported as a flat list. When using the HierarchicalGrid data source, only the top grouping level is exported. • **Icons:** Icons are omitted from the export. • **Column group headers:** Column group headers are omitted from the export. For more information, see Exporting Data Grid contents to a spreadsheet.

Item	Description
Enable Row Selection	Specifies whether users can select a row in the grid. By default this is disabled, which means rows are not selectable in the grid. If enabled, then rows are selectable in the grid. When a user selects a row, the values for that row are written back to the DataGridColumns data source, in the SelectedRowValue column. The form can be configured to change in some way based on the currently selected values. For more information, see Interactive behavior. If the component uses a HierarchicalGrid data source, this option does not display. Instead, this option can be enabled separately for each grouping level within the data source.
Auto Submit	Specifies whether the Axiom form automatically updates when a user selects a row in the grid. This option only applies if **Enable Row Selection** is enabled. By default, auto submit is disabled. You should leave this option disabled if you have not enabled row selection. However, if you have enabled row selection, then in most cases you will want to enable auto submit as well. If both auto submit and row selection are enabled, then the form automatically updates when the user selects a row in the grid (by clicking on it). If auto submit is disabled but row selection is enabled, then the user must use a separate Button component (or a different auto-submit component) in order to update the form for the selected row. If the component uses a HierarchicalGrid data source, this option does not display. Instead, this option can be enabled separately for each grouping level within the data source.

Item	Description
Component Dependencies	Optional. Specifies one or more components that the Data Grid component is dependent on. If you want the data grid to dynamically update based on changes made to other components, list one or more component names in this field. Separate multiple component names with commas. If a component name is listed here, then the data grid is refreshed when a form update submits a change to the listed component. If no component names are listed here, or if the listed components are unchanged, then the data grid is not refreshed when a form update occurs. Components listed as component dependencies must be interactive components, such as Combo Box components, Check Box components, and so on. The purpose of this option is that you want to enable refreshing the data grid based on a change a user made to an interactive component. Non-interactive components, such as Label components, cannot submit values back to the source file and cannot trigger form updates. Therefore, non-interactive components cannot cause the data grid to refresh. **NOTE:** Standard Button components can be used as component dependencies. If a button uses the default Command behavior, then whenever the listed button triggers a form update, the data grid will be refreshed. However, if the button uses a specialized button behavior, or if the button uses a command that alters the normal form update behavior, then the button may not cause the data grid to refresh. For more information, see Update behavior.
Enable Drilling	Optional. Select this check box to enable drilling for the data grid. If enabled, users can "drill down" a row in the grid to see the data in that row at a different level of detail. The remaining properties in this section, such as **Drill Button Tooltip** and **Drilling Hierarchies**, only apply if drilling is enabled. For more information about setting up and using drilling for a Data Grid component, see Setting up drilling for Data Grid components in Axiom forms. If the component uses a HierarchicalGrid data source, this option does not display. Instead, this option can be enabled and configured separately for each grouping level within the data source.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

For Grid components, the grid-level style only impacts the external grid container; it does not affect the internal grid contents.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Update behavior

When the Axiom form is initially rendered, the Data Grid component queries data from the Axiom Software database based on its component settings and its data source settings. This data and the overall grid state (such as visible columns) will remain the same until one of the following occurs:

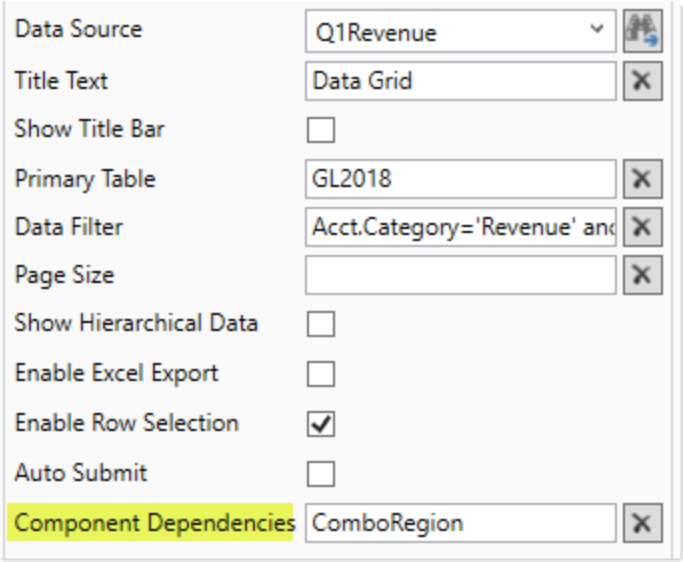
- If the form uses refresh variables, applying changed refresh variables via the Filters panel will refresh the data grid. This means that the data grid can be set up to change its data based on the selected value of a refresh variable.
- If one or more components are listed in the **Component Dependencies** property for the data grid, the data grid is refreshed when a changed value is submitted for one of those components. Otherwise, if no components are listed, or if no changes are submitted for listed components, then form updates triggered by interactive components do *not* cause the data grid to refresh.

By default, when an update is triggered in the form, the data grid is preserved as is. The data query is not run again, the data source is not read again, and any user changes made in the grid are preserved. This behavior is intended to improve performance by not executing the data query and not redrawing the grid every time a form update occurs. It is also intended to improve usability by retaining the user's place in the grid when the grid data remains unchanged.

For example, imagine that the form contains a Combo Box component that is set to auto-submit. When a user selects a value from the combo box, this value is submitted to the source file and a form update is triggered. Under normal circumstances, if another component is configured to dynamically change based on the currently selected value for the combo box, this change would be reflected in the form once the form update is complete. However, the Data Grid component does *not* update in this circumstance. Even if the selected value for the combo box impacts a data grid property—such as the primary table, or the data filter, or the visible columns—the data grid will not change during this form update.

If you want the Data Grid component to update based on the selected value of the Combo Box component, then you must list the name of the Combo Box component in the **Component Dependencies** property for the Data Grid component. For example, if the Combo Box component is

named ComboRegion because it is used to select a region, you would list ComboRegion as a component dependency.



Now when a change is submitted for the Combo Box component named ComboRegion, the Data Grid component is refreshed. The data query is run based on the current component properties and data source properties, and the state of the grid is reset. This occurs at the end of the form update process, when the form display is updated in the browser.

When a form update is triggered, Axiom Software checks to see if any component names are listed in the **Component Dependencies** property of the Data Grid component. You can list multiple component names, separated by commas. If any components are listed, Axiom Software then checks to see if any changes to those components are included in the current form submission. If the listed components are unchanged, the Data Grid component is not refreshed during the form update. If one or more of the listed components are changed, then the Data Grid component is refreshed.

NOTE: The components in Component Dependencies do not have be set to auto-submit in order to refresh the Data Grid component. If an interactive component is changed but it is not configured to auto-submit, then its change will be submitted when the next form update is triggered (either by a Button component, or by a different component that is configured to auto-submit). The Data Grid component will still recognize the component change, even though the change was submitted by a different component.

If a Data Grid component is not refreshed as part of a form update, the following user changes made to the grid are preserved:

- Filters (to columns with `[IsFilterable]` enabled)
- Sorting changes
- Column width and order changes
- Expanded or collapsed groupings

- Scroll state
- Currently selected row
- Current page (when grid results are paged)

When a Data Grid component is refreshed as part of a form update or as part of applying refresh variables, all user changes are lost and the grid is reset. This is necessary because the previous user changes may no longer be relevant, due to the refreshed data.

Interactive behavior

If **Enable Row Selection** is enabled for the Data Grid component, users can select a row in the grid. The values in the selected row are submitted back to the source file, and written to the SelectedRowValue column of the DataGridColumns data source.

If you want the Axiom form to respond to the currently selected row, then you must set up the file so that another component references one or more of the selected values, and changes based on those values. For example, you could have a chart that updates to show information about the department for the currently selected row. For general information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

The row selection feature is the primary means of impacting the form based on user interaction in the grid. However, you can also set up interactive behavior for icons displayed in the grid, to execute a command when a user clicks on an icon. For more information on using actions with icons, see Using the IconConfig data source.

Example

A data grid could display data summed by department:

Dept	Desc	Months	
		Jan	Feb
20000	Corporate	$376,388	$389,632
21000	Corporate Administration	$118,402	$156,274
22000	Information Technologies	$502,486	$521,661
23000	Purchasing & Materials Mgmt	$141,996	$187,932
24000	Business Development	$32,125	$31,618

If a user selects the row for Dept 22000, then the values in that row are submitted back to the source file and written to the SelectedRowValue column in the DataGridColumns data source:

	A	B	C	D	E	F	G
4							
5		[DataGridColumns;Report]	[ColumnName]	[Header]	[IsVisible]	[IsSumBy]	[SelectedRowValue]
6		[Column]	Dept.Dept	Dept	TRUE	TRUE	22000
7		[Column]	Dept.Description	Desc	TRUE		Information Technologies
8		[ColumnGroup]		Months	TRUE		
9		[Column]	GL2017.m1	Jan	TRUE		$502,486
10		[Column]	GL2017.M2	Feb	TRUE		$521,661

There are a number of ways that the form could respond to the selected value in the grid. For example, you might want to display detailed information about the selected row in another grid or in a chart. The grid or chart would need to be set up with formulas that look to the appropriate cells of the SelectedRowValue column, so that the data in the grid or chart changes based on the currently selected value.

If the Data Grid component is grouped, users can still select rows in the grid. In this case, row values are written back for the selected grouping level and all grouping levels above it, but not for any grouping levels below it.

For example, imagine that you have a grid that shows data grouped by Country > Region > Dept. If you select a row in the Country grouping, then the current country value is written back but the region and department values are not. This is because the selected country row does not have an associated region or department. However, if the user expands the groupings down to the lowest level, and then selects the row for Dept 40000, in Region Northwest, in Country United States, then all three dimension values are written back to the data source.

Showing data grouped by dimensions

You can show data in the grid grouped by hierarchical dimensions. Instead of flat rows of data, data is shown in expandable / collapsible groups with nested data. There are two ways to accomplish this:

- You can use the basic option **Show Hierarchical Column Data**. When using this option, data is automatically grouped based on the sum by columns for the grid. All grouping levels use the same columns and the same grid features. This option is the easiest to set up, but it is less flexible in how data is displayed.
- You can use the advanced option to define a HierarchicalGrid data source. When using this option, you define the columns for each grouping level separately, using their own DataGridColumn data sources. You then set up each grouping level in the HierarchicalGrid data source, by specifying the DataGridColumn data source to use for each grouping level, as well as to define the grid properties for each level. This option is more time-consuming to set up, but it allows more flexibility in how data is displayed. For more information, see Using the HierarchicalGrid data source.

To use the basic grouping option:

- In the component properties, enable **Show Hierarchical Column Data**.
- In the data source, define the grouping levels by adding the appropriate reference table columns, and set them to `True` in the `[IsSumBy]` column. The top-level grouping must be listed first, followed by the next level, and so on. There must be at least two sum by columns defined in the data source to create a grouping.

The following example data source shows sum by columns of `Dept.WorldRegion`, `Dept.Region`, and `Dept.Dept` (in that order). This means that WorldRegion is the top-level grouping, and Dept is the lowest level.

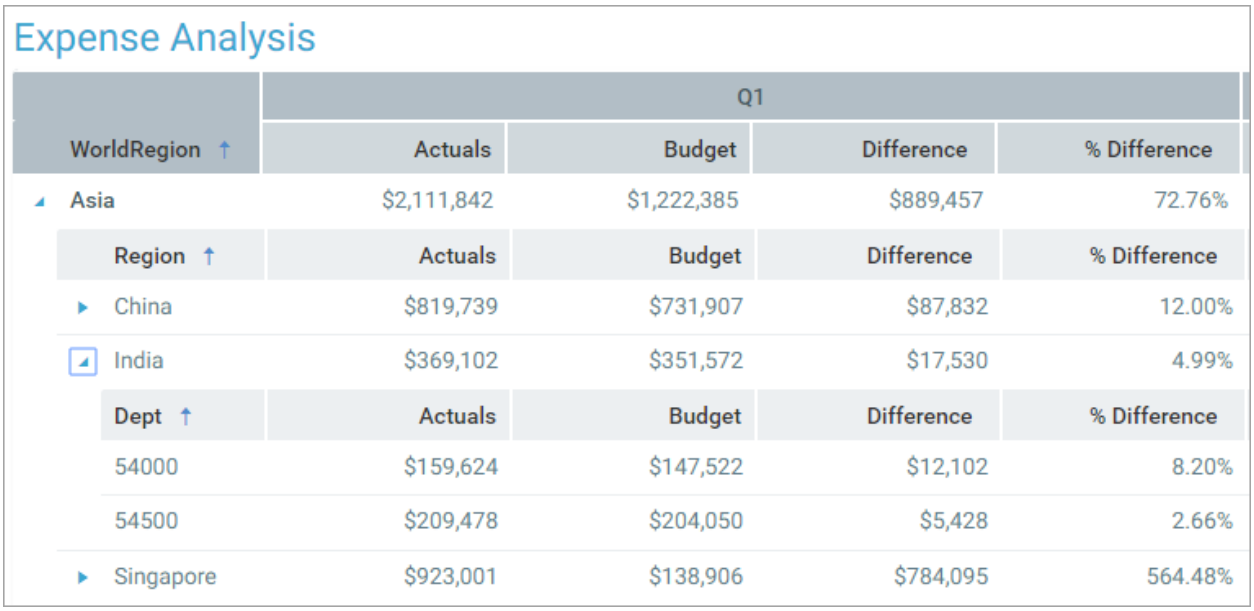
	A	B	C	D	E	F	G	H
4								
5		[DataGridColumns;Q1Revenue]	[ColumnName]	[Header]	[IsVisible]	[SortOrder]	[SortDirection]	[IsSumBy]
6		[Column]	Dept.WorldRegion	WorldRegion	TRUE	1		TRUE
7		[Column]	Dept.Region	Region	TRUE	2		TRUE
8		[Column]	Dept.Dept	Dept	TRUE	3		TRUE

When this grid is rendered, it will display as follows, with data initially grouped by world regions.

Expense Analysis

	Q1			
WorldRegion ↑	Actuals	Budget	Difference	% Difference
▸ Asia	$2,111,842	$1,074,905	$1,036,937	96.47%
▸ Corporate	$7,432,884	$6,710,154	$722,730	10.77%
▸ Europe	$226,016	$89,167	$136,849	153.48%
▸ North America	$15,396,331	$11,351,623	$4,044,708	35.63%

You can expand a world region to see the region data underneath it, and then expand a region to see the department data for that region.



Expense Analysis

WorldRegion	Q1 Actuals	Q1 Budget	Q1 Difference	Q1 % Difference
Asia	$2,111,842	$1,222,385	$889,457	72.76%
Region	Actuals	Budget	Difference	% Difference
China	$819,739	$731,907	$87,832	12.00%
India	$369,102	$351,572	$17,530	4.99%
Dept	Actuals	Budget	Difference	% Difference
54000	$159,624	$147,522	$12,102	8.20%
54500	$209,478	$204,050	$5,428	2.66%
Singapore	$923,001	$138,906	$784,095	564.48%

The other columns in the data source are the same for all levels. For example, if the column `GL2018.Q1` is included in the grid, then you will see that data at the world region level, then the region level, then the department level.

If you want to show descriptions for certain grouping levels, then you must use the DisplayFormat to concatenate the descriptions with the column values. For example, if you want to show descriptions for the Dept level, you can define a display format for the Dept column that is something like `{Dept.Dept} - {Dept.Description}`. This will display both the department code and the description within the Dept column. You cannot simply add Dept.Description as a visible column to the grid in this case, because then it will display for all grouping levels.

NOTES:

- The column width of the first (top-level) sum by column determines the column width for all of the grouping columns. Column widths set for the other sum by columns are ignored. All of the grouping columns are displayed within the same column space, nested underneath each other. Each nested grouping is indented slightly from the parent grouping. Additionally, each parent grouping has to reserve space for the expand / collapse icons shown to the right of the column values. Therefore the first sum by column must have a column width that is wide enough to accommodate all nested levels of groupings. You may need to test various width settings before determining the appropriate column width for the top-level column.
- Frozen columns are not supported with hierarchical data. The `[IsFrozen]` column is ignored.
- If hierarchical data is enabled, all designed sum by columns are automatically visible (regardless of `[IsVisible]`) and automatically displayed in the far left column of the grid (regardless of where they are located in the data source).
- If hierarchical data is enabled, the grid columns are resized as needed so that they fill the entire component width. However, if the columns exceed the component width, the column width values are honored.

Creating column groups for the grid header

You can create column groups in the DataGridColumns data source, for purposes of defining additional header text that spans the column group. This works as follows:

- The row tag `[ColumnGroup]` indicates that you want to start a group. All columns that follow this tag belong to the group, until the group is ended.

  The data source properties of `[Header]` and `[HeaderAlignment]` can be used with `[ColumnGroup]` rows, to indicate the header text for the column group, and to indicate the alignment of that text across the grouped columns. If no alignment is specified, the default is centered. No other data source properties apply, and will be ignored if set.

- The row tag `[EndGroup]` indicates that you want to end a column group. This tag can be omitted if the group extends to the end of the column list. Data source properties do not apply to `[EndGroup]` rows. If an `[EndGroup]` tag cannot be matched to a corresponding [ColumnGroup] tag, an error occurs when rendering the grid.

For example, you may want to define grouped header text such as "Q1" for the columns representing the months of the first quarter. You can place those columns in a column group and define header text for the group using the `[Header]` property. After the last column in the first quarter, you can end the group and then start a new group for Q2.

Groups can be nested for multiple levels of column headers. The `[EndGroup]` tag ends the closest column group to the tag, leaving any other column groups open. If you want to end multiple groups, you must have multiple end tags.

The following example data source shows two levels of column groups. The first, top-level column group indicates the year for the columns (2017). Then multiple, second-level column groups indicate the relevant quarter (Q1, Q2, etc.). In this example, the `[HeaderAlignment]` property is used to align the year header text on the left side of the group, while the header text for the quarters uses the default center alignment.

	A	B	C	D	E	F
4						
5		[DataGridColumns;GridColumnsConfig1]	[ColumnName]	[Header]	[IsVisible]	[HeaderAlignment]
6		[Column]	Dept.Dept	Dept	TRUE	
7		[Column]	Dept.Description	Desc	TRUE	
8		[ColumnGroup]		2017		Left
9		[ColumnGroup]		Q1		
10		[Column]	GL2017.m1	Jan	TRUE	
11		[Column]	GL2017.M2	Feb	TRUE	
12		[Column]	GL2017.M3	Mar	TRUE	
13		[CalculatedColumn]		Q1Total	TRUE	
14		[EndGroup]				
15		[ColumnGroup]		Q2		
16		[Column]	GL2017.M4	Apr	TRUE	
17		[Column]	GL2017.M5	May	TRUE	
18		[Column]	GL2017.M6	Jun	TRUE	
19		[CalculatedColumn]		Q2Total	TRUE	
20		[EndGroup]				
21		[ColumnGroup]		Q3		
22		[Column]	GL2017.M7	Jul	TRUE	
23		[Column]	GL2017.M8	Aug	TRUE	
24		[Column]	GL2017.M9	Sept	TRUE	
25		[CalculatedColumn]		Q3Total	TRUE	
26		[EndGroup]				
27		[ColumnGroup]		Q4		
28		[Column]	GL2017.M10	Oct	TRUE	
29		[Column]	GL2017.M11	Nov	TRUE	
30		[Column]	GL2017.M12	Dec	TRUE	
31		[CalculatedColumn]		Q4Total	TRUE	
32		[EndGroup]				
33		[EndGroup]				
34		[ColumnGroup]		2018		Left
35		[ColumnGroup]		Q1		
36		[Column]	GL2018.m1	Jan	TRUE	

Example data source with nested groups

The first end tag in row 14 ends the closest column group, which is the Q1 group. The next tag in the data source starts the Q2 column group. The 2017 column group is left open to continue to span over all of

the quarter groups. When the end of the Q4 group is reached, there are two end tags to end both the Q4 group and the 2017 group (rows 32 and 33). Then the groups start again with 2018 and its first quarter.

When this grid is rendered, the headers look as follows:

		2017							
		Q1				Q2			
Dept ↑	Desc	Jan	Feb	Mar	Q1Total	Apr	May	Jun	Q2Total
45500	San Francisco - Store 87	$96,673	$76,642	$51,006	$224,320	$74,832	$51,006	$74,832	$200,671
46000	Chicago - Store 45	$388,804	$506,292	$508,908	$1,404,003	$633,748	$518,626	$697,332	$1,849,706
47000	Portland - Store 94	$314,682	$389,239	$582,960	$1,286,881	$475,948	$591,103	$796,278	$1,863,328

NOTES:

- Header text for a column group is optional. You may want to use column groups with no header text as "spacer" rows, to accommodate headers with varying levels of groupings.
- Column groups cannot be used with frozen columns. If a column belongs to a column group but is also flagged as a frozen column, the frozen status is ignored. It is not possible to define column groups within the frozen columns area.
- If **Show Hierarchical Column Data** is enabled, column groups cannot be used with the sum by columns. If a sum by column is part of a column group, this will be ignored and the column will not show underneath the group header.

Design alternatives

The Data Grid component is designed to handle a very specific purpose, to display reporting data. If you need more flexibility, you can use a Formatted Grid component. For example, you might use a Formatted Grid component in the following cases:

- When you need to use more advanced data query configurations or features. You can use Axiom queries to bring data into the source file, and then use the Formatted Grid component to display that data in the form.
- When you need users to be able to edit data in the grid and save the changed data to the database. Formatted Grid components support many options for editing data.
- When you need the ability to format the grid contents at a more granular level. Formatted Grid components support a variety of formatting options to format grid contents, including fonts, colors, and borders.
- When you need the ability to print the form as a PDF. Formatted Grid components support the ability to print all rows by using the Extended Height option. Data Grid components currently do not support the ability to print to PDF.

Using the HierarchicalGrid data source

You can optionally use a HierarchicalGrid data source with a Data Grid component, in order to show data in the grid grouped by dimensions. This data source defines a list of DataGridColumns data sources, so that each grouping level can be defined as a separate grid with its own unique primary table, columns, and other properties.

This feature is an alternative to using the built-in Show Hierarchical Column Data option in the Data Grid properties. You should use a HierarchicalGrid data source instead if:

- You need different grouping levels to show different columns. When using a HierarchicalGrid data source, each grouping level has its own DataGridColumns data source to determine the columns that display for that level. When using the Show Hierarchical Column Data option, all levels use the same DataGridColumns data source and show the same columns.
- You need different grouping levels to use different primary tables and/or filters. When using a HierarchicalGrid data source, each grouping level specifies its own primary table and filter, so each level can query a different set of data. When using the Show Hierarchical Column Data option, all levels use the same primary table and filter.
- You need different grouping levels to use different grid options, such as enabling drilling or the ability to select rows. When using a HierarchicalGrid data source, each grouping level has its own defined set of grid options that apply only to that level. When using the Show Hierarchical Column Data option, all levels use the same grid options.

The basic setup steps to use a HierarchicalGrid source are as follows:

- Create a DataGridColumns data source for each grouping level that you want to show in the grid.
- Create a HierarchicalGrid data source and set up a row for each DataGridColumns data source.
- Place a Data Grid component on the form canvas, and select the HierarchicalGrid data source as the data source for the component.

Defining a HierarchicalGrid data source

The tags for a HierarchicalGrid source are as follows:

Primary tag

[HierarchicalGrid;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Data Grid component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Grid]

Each row flagged with this tag specifies a separately defined DataGridColumns data source to show as a grouping level in the Data Grid component. The first Grid row is the top-level grouping, the second grid row is the second level grouping, and so on.

Column tags

[GridColumnsDataSource]

The name of the DataGridColumns data source that defines the columns to display for this grouping level in the grid. Enter the name of any DataGridColumns data source.

The DataGridColumns data sources referenced here can use any of the normal features of DataGridColumns data sources, including enabling filters for columns, applying alternate aggregation or column filters, and using icons.

[PrimaryTable]

The name of the primary table for the grouping level. Enter any valid table name from the Table Library. The primary table determines which table columns are valid to include in the DataGridColumns data source for the grouping level. System tables (such as Axiom.Columns) cannot be used as the primary table.

For example, if you specify GL2018 as the primary table, then the query can retrieve data from that table, plus any reference tables that the primary table looks up to. If you want to include data from multiple data tables, you can include any table that shares keys with the primary table, as well as any shared lookup reference tables.

Each Grid row can use a different primary table, however, each grouping level must be filterable by the levels above it. Generally speaking, this means that all of the primary tables must share a common lookup column, and the sum by columns for each grouping level must be compatible with that lookup column. For more information, see How the data in each grouping level is determined.

[Filter]

Optional. A filter to limit the data returned by the query and displayed in the grouping level. Enter a filter criteria statement that is valid in the context of the primary table. If no filter is defined, all data that matches the query is displayed in the grouping level.

[PageSize]

Optional. A number that determines how many rows are shown in each "page" of the grouping level. If the results returned by the query exceed the page size, then the data is separated into multiple pages. Users can use the page controls at the bottom of the grouping to move among pages.

If omitted or blank, the default page size is 50. If set to 0, then all rows display in each grouping, with no paging.

[EnableRowSelection]

Optional. Specifies whether users can select a row in the grouping level (True/False). If False or omitted, then rows in the grouping level are not selectable.

If True, then rows are selectable in the grouping level. When a user selects a row, the values for that row are written back to the corresponding DataGridColumns data source, in the SelectedRowValue column. The form can be configured to change in some way based on the currently selected values.

For general information on how row selection works for Data Grid components, see Interactive behavior. Note the following when using row selection with a HierarchicalGrid data source:

- Although multiple grouping levels can be enabled for row selection, only one row across all groupings can be the currently selected row at any one time.
- When a row is selected in a grouping, values are written back for that row as well as for all grouping levels above that row. For example, if you expanded Region West and then selected Dept 42000, values are written back for the Region West row as well as the selected Dept 42000 row, in the corresponding DataGridColumns data sources.
- In addition to the row values, the grouping filter corresponding to the selected row is written back to the SelectedRowFilter column in the HierarchicalGrid data source.

[AutoSubmit]

Optional. Specifies whether the Axiom form automatically updates when a user selects a row in the grouping level (True/False). This option only applies if EnableRowSelection is True.

If False or omitted, auto submit is disabled. You should leave this option disabled if you have not enabled row selection. However, if you have enabled row selection, then in most cases you will want to enable auto submit as well.

If both auto submit and row selection are enabled, then the form automatically updates when the user selects a row in the grouping level (by clicking on it). If auto submit is disabled but row selection is enabled, then the user must use a separate Button component in order to update the form for the selected row.

[EnableDrilling]

Optional. Specifies whether drilling is enabled for rows in the grouping level. If False or omitted, then drilling is not enabled for the grouping level. If enabled, users can "drill down" a row in the grouping level to see the data in that row at a different level of detail.

For more information about how drilling works for Data Grid components, see Setting up drilling for Data Grid components in Axiom forms. Drilling works the same way when using a HierarchicalGrid data source; the only difference is that the drilling options are configured in the HierarchicalGrid data source instead of in the component properties.

[DrillButtonTooltip]

Optional. Defines text to display in a tooltip when a user hovers their cursor over the drill icon. If left blank, no tooltip displays on hover.

[DrillHierarchy]

Optional. Specify one or more hierarchies to determine the drilling levels available to users. For more information on how to specify the desired hierarchies, and how users select from the hierarchy levels, see Using hierarchies to define drilling levels.

[DrillLevelsDataSource]

Optional. Enter the name of a DrillLevels data source. If specified, users will be presented with the custom drilling options defined in this data source. For more information on creating the data source, and how users select from the custom drilling options, see Using a DrillLevels data source to define drilling levels.

[SelectedRowFilter]

System-controlled field. If a row is selected in the grid, this field is populated with a filter that represents the dimension for the selected row. This field is automatically updated by Axiom Software when a user selects a row in the grid.

For example, if a user selects the row for Dept 42000, then this field is updated to contain the filter Dept.Dept=42000 (on the Grid row corresponding to the Dept grouping level). If the selected row is within a grouping level that is not the top-level grouping, then the corresponding filters are also written for all grouping levels above the selected row.

This filter is for reference only and can be used to drive other components in the form if desired.

[ActionRowFilter]

System-controlled field. If an icon in the grid is used to trigger an action, this field is populated with a filter that represents the dimension for the row with the icon. This field is automatically updated by Axiom Software when a user clicks an interactive icon in the grid.

For example, if a user clicks the icon in the row for Dept 42000, then this field is updated to contain the filter Dept.Dept=42000 (on the Grid row corresponding to the Dept grouping level). If the row is within a grouping level that is not the top-level grouping, then the corresponding filters are also written for all grouping levels above the selected row.

This filter is for reference only and can be used to drive other components in the form if desired.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

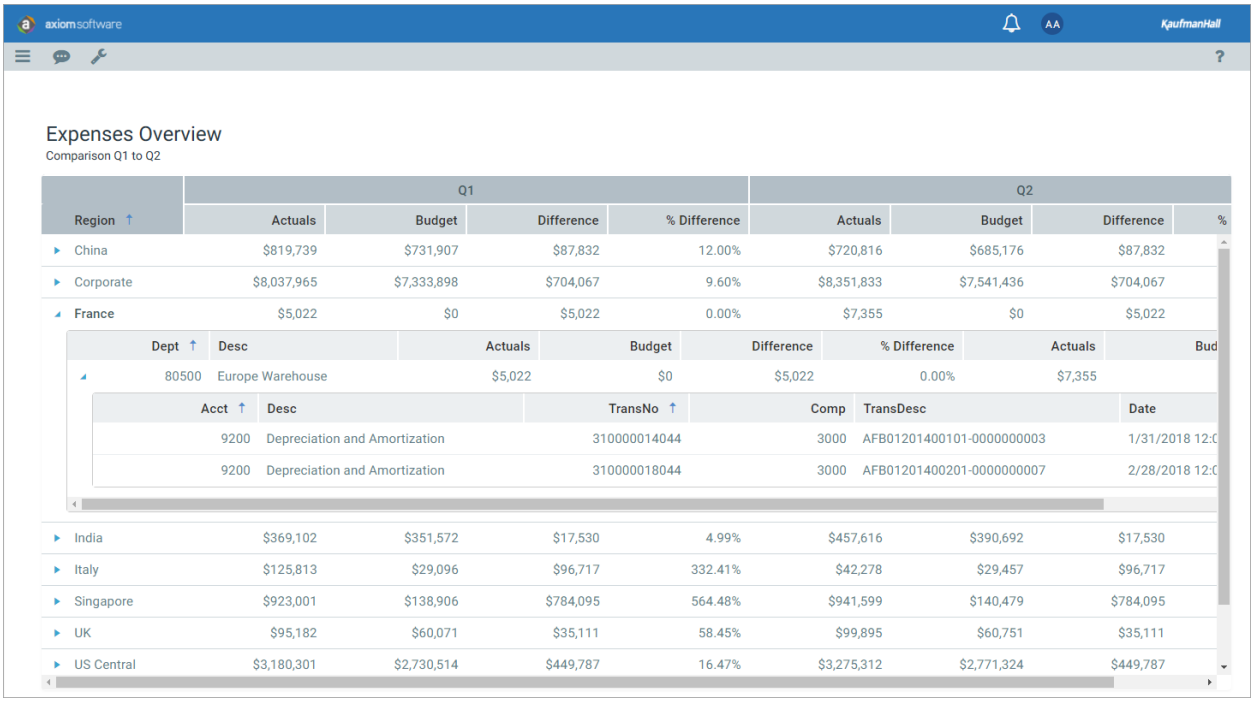
To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Hierarchical Data Grid**. You can also highlight a range of data first and then use the wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The following example shows a sample HierarchicalGrid data source tagged in a sheet:

	C	D	E	F	G	H	I	J	K
3									
4	[HierarchicalGrid;RegionalGroup]	[GridColumnsDataSource]	[PrimaryTable]	[Filter]	[PageSize]	[EnableRowSelection]	[AutoSubmit]	[EnableDrilling]	[DrillButtonTooltip]
5	[Grid]	Region	GL2018	Acct.AcctCat='Expense'					
6	[Grid]	Dept	GL2018	Acct.AcctCat='Expense'	5	TRUE	TRUE		
7	[Grid]	GLDetail	GLDetail2018	Acct.AcctCat='Expense'	10				

Example HierarchicalGrid data source

The resulting data grid for the example data source shown above looks as follows:

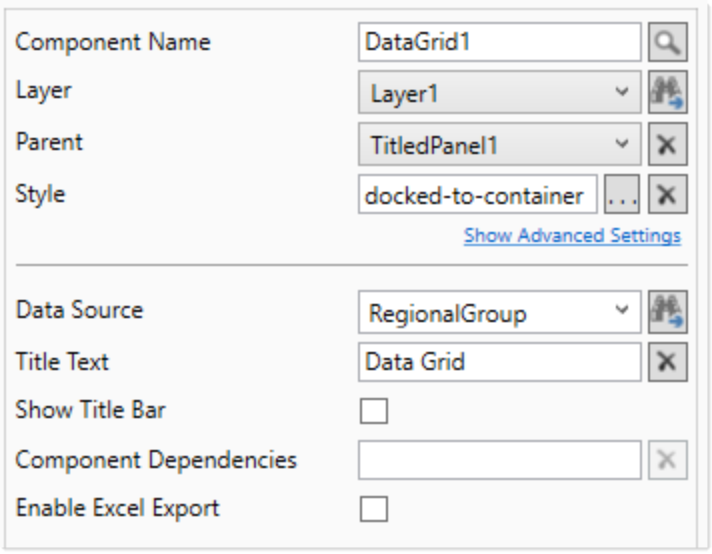


Example data grid using HierarchicalGrid data source

In this example, the first two grouping levels are showing the same basic columns and querying the same table, just using a different sum by level. The third level queries a different table and shows a completely different set of columns.

Configuring a Data Grid component to use a HierarchicalGrid data source

When you configure the **Data Source** for a Data Grid component, you can select either a regular DataGridColumns data source or a HierarchicalGrid data source. When you select a HierarchicalGrid data source, most of the component-level properties become hidden because they are no longer set at the component level.



Example component properties when a HierarchicalGrid data source is selected

Notice that settings such as Primary Table, Data Filter, and Enable Row Selection no longer display in the component properties. Instead, these properties are set individually for each grouping level, within the HierarchicalGrid data source.

If any of the hidden properties are configured on the Form Control Sheet, they will be ignored when the data source is a HierarchicalGrid data source.

How the data in each grouping level is determined

When the Data Grid component is rendered in the form, it reads the grouping levels from the HierarchicalGrid data source. Essentially, each row in the data source is treated as a separate Data Grid component, with its own component properties and DataGridColumns data source. The top-level grid is the first row of the HierarchicalData data source. If you expand an item in this top-level grid, it displays a child grid based on the second row of the HierarchicalData data source, and so on.

When you expand a grouping to see the next grouping level underneath it, the data is filtered to only show the relevant rows for the expanded grouping. For example, if you have levels of Region > Dept, then when you expand region US West to see the Dept grouping level, you only see the departments that belong to that region.

This is accomplished by applying a filter to the child grid for the second grouping level, based on the sum by column of the first grouping level. In this example, the `Dept.Region` column is the sum by column for the Region grouping level. When region US West is expanded, a filter of `Dept.Region='US West'` is applied to the child grid for the Dept level, in addition to any filter defined for the Dept row in the HierarchicalGrid data source.

This filter is compounded as you move down levels. For example, imagine that you have levels of WorldRegion > Region > Dept. You expand world region North America and then region US West to see

the Dept level. Now the filter applied to the Dept child grid is `Dept.WorldRegion='North America' and Dept.Region='US West'`.

This means that all of the grouping levels in your HierarchicalGrid data source must use compatible primary tables and use sum by columns that can be applied as filters to the lower grouping levels. If the sum by level of a parent grid cannot be applied as a filter to the primary table of a child grid, then an error will occur when the user attempts to expand the grouping level to see the child grid.

Setting up drilling for Data Grid components in Axiom forms

You can enable drilling for Data Grid components and configure the grid so that certain drilling selections are available to users. If drilling is enabled, users can drill any row in the grid by clicking on a drill icon that displays on each row. Users can select a drilling level from among the available selections, and then the drilling results are presented in a separate web page. Users can continue to drill the drilling results if desired, or return to the original grid and drill again from there.

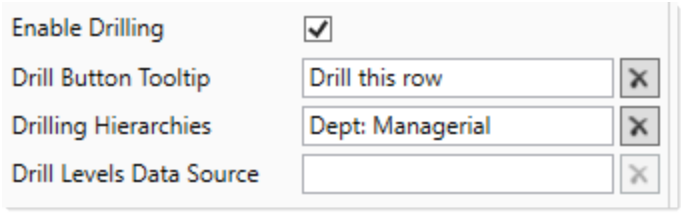
When configuring drilling for the grid, you specify which drilling levels are available to users. You can choose to present users with predefined hierarchies for drilling, or you can define custom drilling levels using a DrillLevels data source.

NOTE: Some browsers may require pop-ups to be allowed for the Axiom Software site in order to perform drilling in a web report.

Enabling drilling for a Data Grid component

To enable drilling for a Data Grid, you must complete the following component properties in the Form Assistant or the Form Designer:

Item	Description
Enable Drilling	Select this check box to enable drilling for the data grid. If enabled, users can "drill down" a row in the grid to see the data in that row at a different level of detail.
Drill Button Tooltip	Optional. Defines text to display in a tooltip when a user hovers their cursor over the drill icon. If left blank, no tooltip displays on hover.
Drilling Hierarchies	Optional. Specify one or more hierarchies to determine the drilling levels available to users. For more information on how to specify the desired hierarchies, and how users select from the hierarchy levels, see Using hierarchies to define drilling levels.
Drill Levels Data Source	Optional. Enter the name of a `[DrillLevels]` data source. If specified, users will be presented with the custom drilling options defined in this data source. For more information on creating the data source, and how users select from the custom drilling options, see Using a DrillLevels data source to define drilling levels.



Example component properties to enable drilling

NOTE: If you are using a HierarchicalGrid data source, then the drilling options are defined for each grouping level within that data source, instead of in the component properties. The same drilling options are available and work the same way.

The drilling options presented to users are determined as follows:

- When drilling the data grid, Drill Levels Data Source is always used if defined, otherwise Drilling Hierarchies is used. If Drilling Hierarchies is blank, all relevant hierarchies are used (based on the primary table).
- When drilling the drill results, Drilling Hierarchies is always used (Drill Levels Data Source is ignored if set). If Drilling Hierarchies is blank, all relevant hierarchies are used (based on the primary table).

IMPORTANT: It is up to the form designer to ensure that any listed hierarchies or custom drill levels are valid in the context of the data displayed in the data grid. If invalid selections are presented to users, errors may occur when drilling.

Using hierarchies to define drilling levels

You can configure the drill so that users select a drilling level from one or more hierarchies that are associated with the grid data. This is similar to how drill-down drilling works for spreadsheet Axiom files in the Desktop Client. However, when using hierarchies to drill in an Axiom form, you can specify which hierarchies you want to be available to the user.

If you want to use hierarchies to define the drilling level, complete the **Drilling Hierarchies** property using one of the following options:

Hierarchy Option	Description	Example
<Blank>	If Drilling Hierarchies is left blank, then all relevant hierarchies are shown to the user (based on the primary table of the data grid being drilled).	N/A

Hierarchy Option	Description	Example

TableName	Enter a table name to display all hierarchies defined for that table.
	
	You can also enter multiple table names, separated by commas. The dialog will display all hierarchies defined for all listed tables.
		`Dept`
		Displays all hierarchies defined on the Dept table.
		
		`Dept,Acct`
		Displays all hierarchies defined on the Dept table and the Acct table.

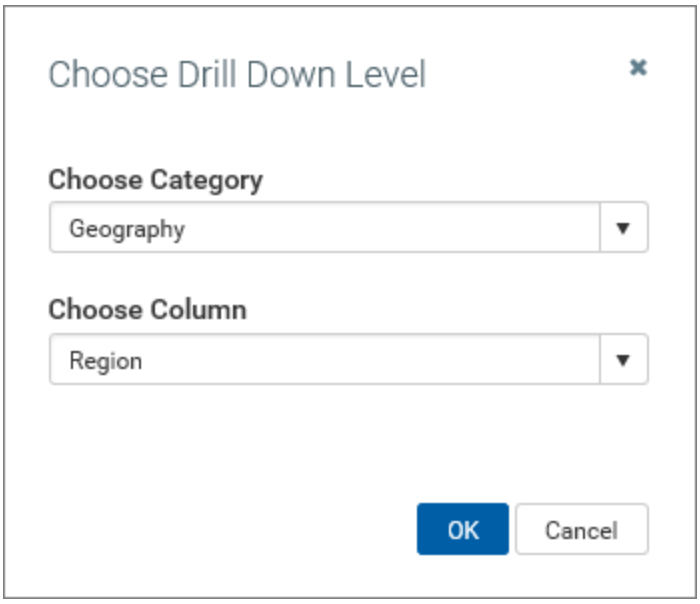
TableName:HierarchyName	Enter a table name plus a hierarchy name to only show the specified hierarchy.
	
	You can also enter multiple table:hierarchy pairs, separated by commas. The dialog will display all specified hierarchies.
		`Dept:Geography`
		Displays the Geography hierarchy defined on the Dept table.
		
		`Dept:Geography, Acct:Type`
		Displays the Geography hierarchy defined on the Dept table and the Type hierarchy defined on the Acct table.

TableName.ColumnName: HierarchyName	Enter a Table.Column name plus a hierarchy name to only show the specified hierarchy path, and to apply the selected hierarchy level in the context of the specified Table.Column. This may be helpful when the query data contains multiple paths to the hierarchy columns, which by default causes hierarchies to show multiple times.	`Dept.Region:Region` Displays the Region hierarchy on the Region table, where `Dept.Region` looks up to the Region table. Additionally, in this example the resulting drilling level will be defined as `Dept.Region.RegionType` instead of just `Region.RegionType` (where RegionType is a level in the Region hierarchy).

NOTE: If the query data contains multiple paths to the hierarchy columns, the same hierarchy will show multiple times (once for each valid path). The Table.Column option is available if you want the hierarchy to always use a particular path, and therefore only that path will be listed.

If you configure the drill to use specific hierarchies, you must make sure that hierarchy is valid within the context of the data grid. The hierarchy must be on a lookup reference table for the primary table of the grid. Additionally, if the grid columns are from multiple data tables, then the hierarchy must be on a shared lookup reference table for all of the data tables in the query.

When hierarchies are used, users first select a category (the hierarchy) and then select a column in the hierarchy. In the following example, the user has selected the Geography hierarchy and then the Region column, so the drilling data will use regions as the rows.



If only one hierarchy is available, then the user does not have to select the hierarchy. Instead, the columns in the hierarchy are presented in the same way as the options from a DrillLevels data source (as shown in the next section).

For more information on creating hierarchies, see the *System Administration Guide*.

Using a DrillLevels data source to define drilling levels

You can configure the drill so that users select from a list of custom drilling choices. Each choice corresponds to a table column that you want to allow users to drill by. This provides you with full control over how the drilling levels are presented and which columns can be used to drill.

To define custom drilling choices:

- Create a `[DrillLevels]` data source on any sheet within the form-enabled file. The data source defines the columns that can be used to drill, and the display text to show to users for each option.
- Enter the data source name in the **Drill Levels Data Source** field in the Data Grid component properties.

The tags for the `[DrillLevels]` data source are as follows:

Primary tag

[DrillLevels;*DataSourceName*]

The DataSourceName uniquely identifies this data source so that it can be assigned to a Data Grid component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[DrillItem]

Each row flagged with this tag specifies a drilling option to present in the Drill Level selection dialog.

Column tags

[Label]

Defines the display name of each item in the Drill Level selection dialog. This label is also used as the column header in the drill results.

[Column]

Defines the Table.Column to use for the drilling level when the corresponding label is selected. For example, if the column is Dept.Region, then the drilling data is by region. Multiple-level lookups can be used.

It is up to the form designer to ensure that each column listed is valid and relevant to the data that can be drilled. Generally speaking, if the data query only uses one data table, then any column in the table itself as well as any column in lookup reference tables can be used. If the data query uses multiple data tables, then only shared lookup reference tables can be used. Other columns may return unexpected drilling data, or may result in drilling errors.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

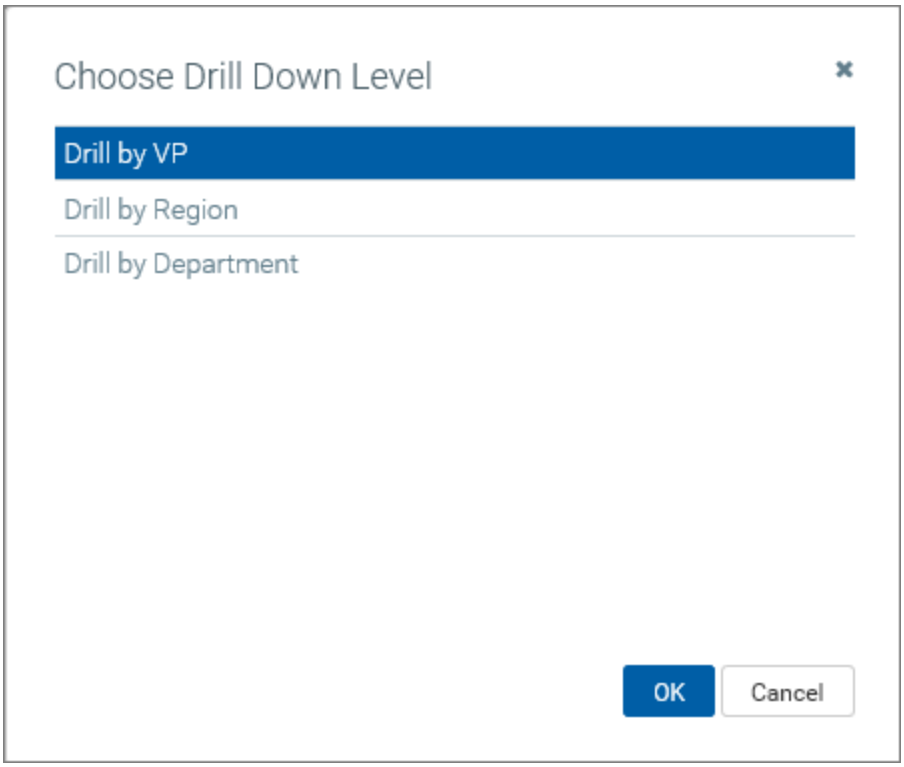
To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Drill Levels**. You can also highlight a range of data first and then use the

wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The following example shows a DrillLevels data source:

	A	B	C	D	E
6					
7			[DrillLevels;DeptDrill]	[label]	[column]
8			[DrillItem]	Drill by VP	Dept.VP
9			[DrillItem]	Drill by Region	Dept.Region
10			[DrillItem]	Drill by Department	Dept.Dept

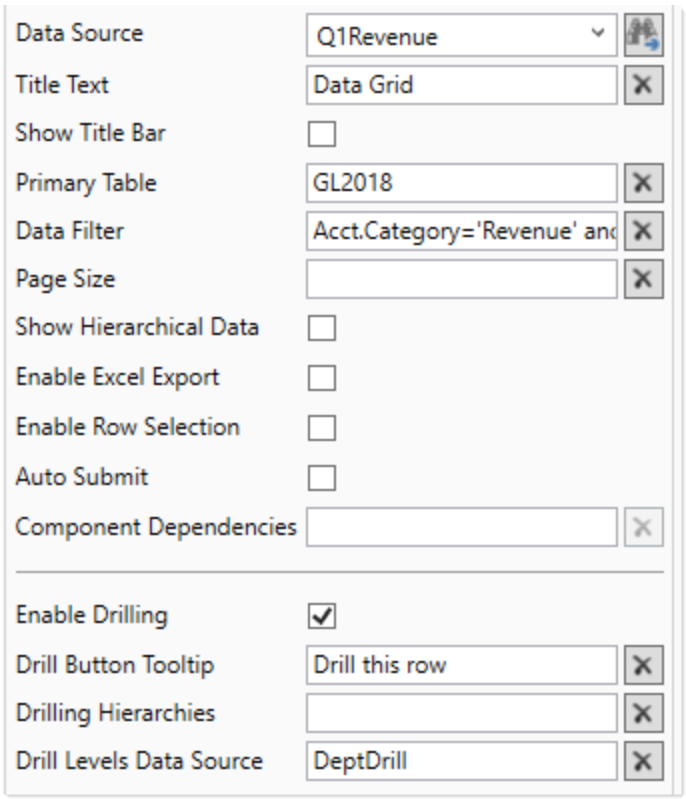
When a user initiates a drill, the drilling items are displayed in the **Choose Drill Down Level** dialog as shown in the following screenshot. Only the label displays; the column is not shown (unless you include the column name in the label).



The drill results are then created using the corresponding column for the selected label.

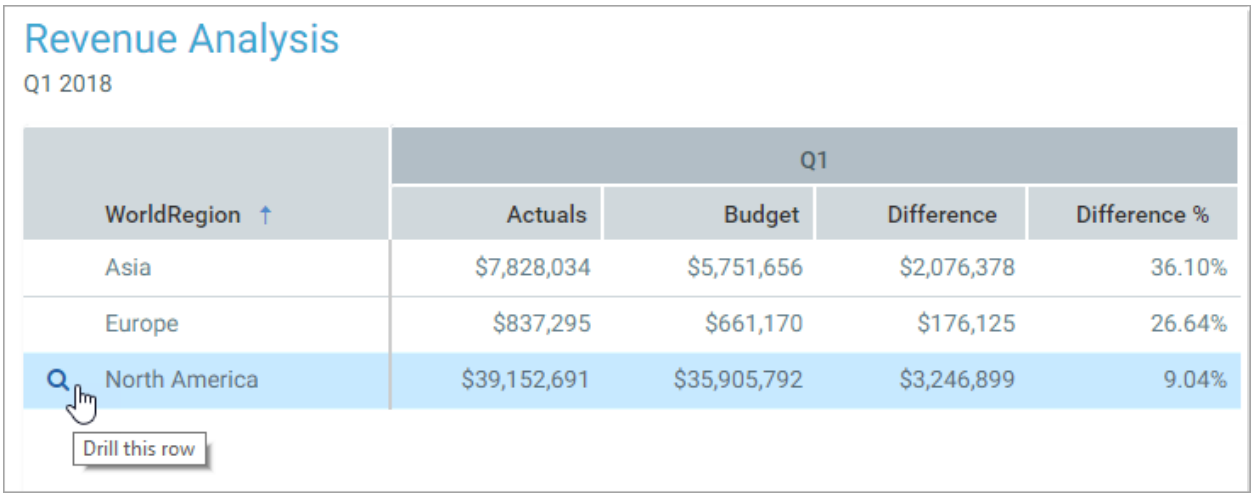
Drilling example

The following example is intended to give form designers an idea of the user experience when drilling a Data Grid component. In this example, the Data Grid component is configured as follows:

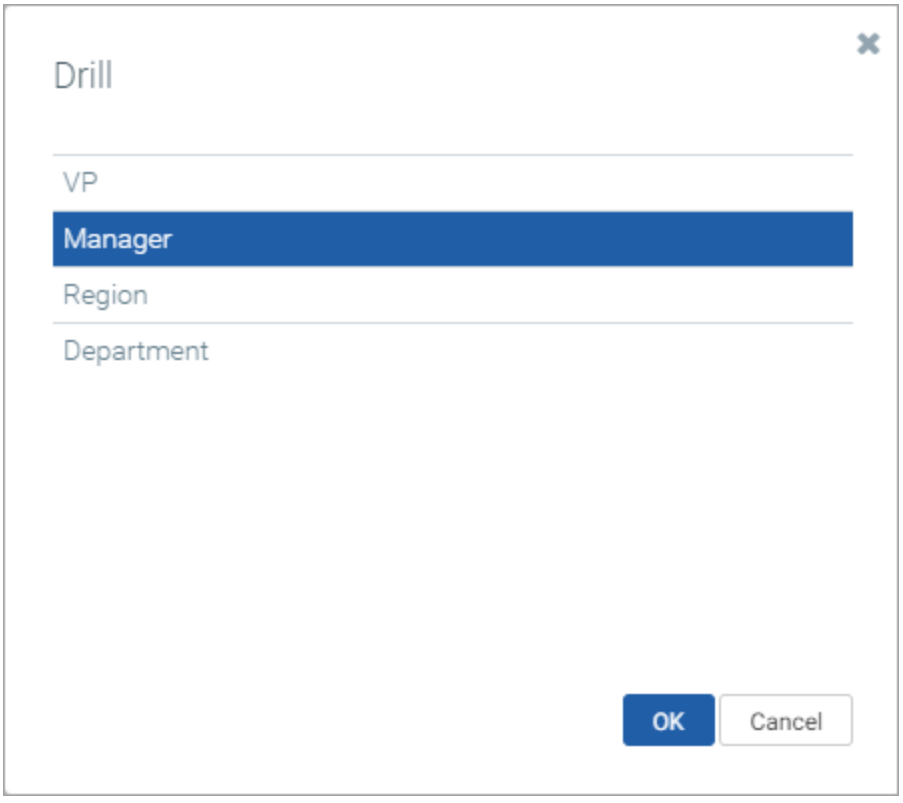


- The primary table is GL2018, which means than any hierarchies or drilling levels must be valid in the context of this table.
- **Enable Drilling** has been enabled for the component.
- The **Drilling Hierarchies** property is left blank, but a data source name is defined for **Drill Levels Data Source**. This means that the data source will be used to define the drilling options for the data grid instead of using a hierarchy.

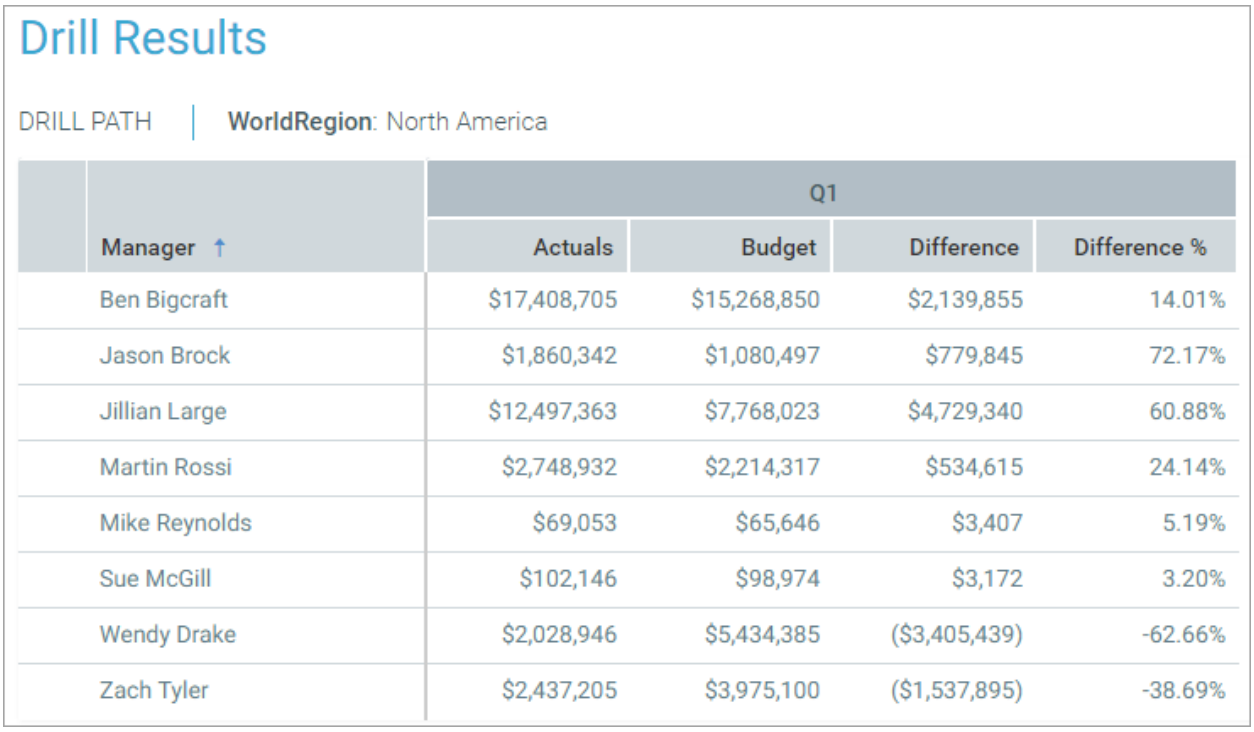
When this file is viewed as a form, the first column in the data grid is now the drill action column. When a user hovers their cursor in that column, they can see a drill icon (a magnifying glass) for the current row. The user can click on the icon to initiate a drill for that row.



Once the user has initiated the drill, a dialog opens to display the available drilling levels. In this example, these are the drilling levels defined in the DrillLevels data source named DeptDrill. (If instead one or more hierarchy names had been specified in the Data Grid properties, the dialog would prompt users to select from the hierarchy levels.)



After the user selects a drill level (Manager in this case), a new browser tab opens to display the drill results. In this example, the data for the North America row is now shown at the Manager level.



Drill Results

DRILL PATH | **WorldRegion**: North America

Manager	Q1 Actuals	Q1 Budget	Q1 Difference	Q1 Difference %
Ben Bigcraft	$17,408,705	$15,268,850	$2,139,855	14.01%
Jason Brock	$1,860,342	$1,080,497	$779,845	72.17%
Jillian Large	$12,497,363	$7,768,023	$4,729,340	60.88%
Martin Rossi	$2,748,932	$2,214,317	$534,615	24.14%
Mike Reynolds	$69,053	$65,646	$3,407	5.19%
Sue McGill	$102,146	$98,974	$3,172	3.20%
Wendy Drake	$2,028,946	$5,434,385	($3,405,439)	-62.66%
Zach Tyler	$2,437,205	$3,975,100	($1,537,895)	-38.69%

The drill results automatically include all columns from the original data grid except for the sum by columns, which are replaced by the selected drill level. The current drilling level is displayed at the top of the page.

NOTE: If the original data grid included a description column, this column will also be included in the drill results but will not be updated for the drill level. You can work around this by using the DisplayFormat property in the DataGridColumns data source, to display descriptions in the same column as their corresponding codes. This does not solve the issue of displaying descriptions for the drill level (currently this is not possible), but it will prevent descriptions from the original data grid from displaying in the drill results.

If desired, the user can further drill on the drill results. Remember when drilling the drill results, the Drill Levels Data Source is ignored and instead the Drilling Hierarchies are used to determine the drill options. Because Drilling Hierarchies was left blank in our grid configuration, the user is presented with all relevant hierarchies for the primary table. If we wanted to limit the drilling options available from the drill results, then we could complete the Drilling Hierarchies property as appropriate and it would be used by the drill results.

When drilling the drill results, the results are presented in the same page, overwriting the current results. A new tab is not opened.

Using the IconConfig data source

Data Grid components support the ability to display icons in the grid. You can create an IconConfig data source in order to define a list of icons and their corresponding colors, conditions, and actions. You can then reference the data source in either the Icon column or the HoverActions column of the DataGridColumns data source.

Creating the data source

The tags for the IconConfig data source are as follows:

Primary tag

`[IconConfig;DataSourceName]`

The DataSourceName identifies this data source so that it can be used in a DataGridColumns data source. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[Icon]`

Each row flagged with this tag defines an icon to display in the data grid.

Column tags

`[IconName]`

The name of an icon to display in the data grid. The valid icon names are the same names allowed for symbols in Formatted Grid components (as well as Label and Button components). You can use any of these features to look up the desired icon name.

> **TIP:** You can right-click the cell and use **Insert Formatted Grid Tag > Symbol**, then use the Tag Editor to select a symbol name (such as `fa-file-o` for a file symbol). You can then copy and paste the symbol name out of the Tag Editor and into the `[Icon]` column.

[Color]

Optional. A color for the icon. You can specify the color using a web-standard color name or a hexadecimal color code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

You can also use an Axiom style color, such as P6 or A32. The skin of the Axiom form must be set to **Axiom2018** in order to use the style colors.

Colors only apply when the data source is used in the Icon column of the DataGridColumns data source. Colors are not supported for use with hover icons.

[Condition]

Optional. A condition to determine whether the icon displays. The condition is evaluated per row in the grid.

For example, imagine that you want to display a green circle to indicate a value that is within acceptable parameters, and a red circle to indicate an unacceptable value. You can set up two different icon rows in the data source (one with the icon set to green and the other with the icon set to red), and set a different condition for each icon so that each row in the grid will evaluate to either green or red. One condition could be `Difference <= 1000` and the other `Difference > 1000` (where Difference is the name of a calculated column in the grid).

The condition can use any of the operators that are valid for use in grid calculations, as well as: greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=), equals (=), and does not equal (<>). You can use AND or OR to create compound statements.

The condition can reference any column that a grid calculation can reference. It can also reference the following:

- Grid calculated columns using the name defined in the ColumnName field. For example, if you name the calculated column `Difference`, then you can reference it using that name in a condition.
- Grid columns with a defined aggregation type or a defined column filter. In this case, you must define a unique name for the column in the ColumnName field using special syntax: *`Table.Column;Name`*. For example: `GL2018.TOT;Filter_Europe` (where the column filter limits the column to data from the Europe world region). You can then reference Filter_Europe as the column name in a condition.

[Action]

Optional. An action to perform when a user clicks on the icon. This column can contain either of the following:

- A URL (starting with HTTP/S) to open a web page, Axiom form, or web report.
- A document shortcut (`document://filepath`) to a file in the Axiom Software file system.
- A command to execute when the icon is clicked. For example, you can use the Dialog Panel command to open a dialog with more information about the current row.

  You can use any command that is available for use in forms, though, some commands may not make sense to execute in this context and may not work as expected. To create the command statement, right-click the cell and select **Axiom Wizards > Command Wizard**. You can then use the Shortcut Target to select a command from the Command Library and configure its shortcut properties, just like you would for a Button component.

For more information on using icons to perform dynamic actions, see Using the current row value in the icon action.

[Confirmation]

Optional. A confirmation message to display before performing the assigned action. The user can click **OK** to continue, or **Cancel** to cancel the action and return to the file.

[UseNewWindow]

Optional. Specifies whether a URL action is opened in a new window (True/False). If omitted or blank, True is assumed.

[Tooltip]

Optional. Tooltip text to display when the user hovers their cursor over the icon. If the icon has an action, this text should tell the user what action is going to be performed.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

Using the current row value in the icon action

You may want to reference the current row value when performing an icon action, so that the action dynamically adjusts for the current row. The way that you do this depends on whether the row action is a URL or a command from the Command Library.

URL actions

If the action is a URL, you can optionally reference the current value of the cell by using the syntax `{value}` in the URL. This means that the icons must be displayed in the same column that holds the value that you want to reference.

For example, imagine that you want users to be able to launch the Process Routing page for each department listed in the data grid, so that they can see the current process status for that department. You can assign the icon data source to the column that shows `Dept.Dept`, and reference the `{value}` in the URL as follows:

```
https://mycompany.axiom.cloud/process/16682/planfile?planvalue={value}
```

This value will be resolved for each row of the grid so that the URL references the appropriate value. For example, if the department value in a row is 40000, the URL will be resolved as follows:
`https://mycompany.axiom.cloud/process/16682/planfile?planvalue=40000`

Command actions

If the action is a command, you can optionally reference any dimension value for the current row, by using the ActionRowValue column in the DataGridColumns data source.

When the user clicks on an icon to perform an action, the dimension values for the current row are written back to the ActionRowValue column. The dimension values are based on the sum by columns for the grid. For example, imagine the sum by columns are `Dept.Dept` and `Acct.Acct`, and a row contains the values Dept 40000 and Acct 2000. When a user clicks on an icon in that row, the values 40000 and 2000 are written back to the ActionRowValue column, in the rows corresponding to `Dept.Dept` and `Acct.Acct`. These values can then be referenced by the command's shortcut parameters directly, or by something else that the command impacts.

For example, imagine that you want to use the Dialog Panel command to open a dialog that shows more information about the department for the current row. In this case, the Dialog Panel command itself does not need to be dynamic—it can be configured to open the same panel. Instead, the child components of the Dialog Panel and the relevant data queries in the form can reference the value in the ActionRowValue column, so that the labels and data in the dialog are dynamically updated for the current department.

If you need to reference the current row value in the command's shortcut parameters directly, this is only possible if the parameter supports bracketed cell reference syntax to read the value from the spreadsheet. For example, you can configure a parameter to use `[Report!D24]`, where that is the ActionRowValue cell for the `Dept.Dept` column in the grid. When the user clicks on the icon, the current department value would be written back to Report!D24 and used in the command's shortcut parameters.

NOTE: It is not supported to construct the command string using a formula, because the formula will not be recalculated when the icon is clicked. The only way to dynamically reference a value in the command's shortcut parameters is to use the bracketed cell reference syntax to read the value from a designated cell.

The ActionRowValue column is only populated when:

- The icon action is a command.
- The command triggers a form update. Most commands do this, but some don't. For example, the File Attachment command does not trigger a form update, so you cannot use that command on an icon and expect to open the File Attachment dialog for the current row value. Make sure to review the form update behavior for the command that you want to use.

Exporting Data Grid contents to a spreadsheet

You can set up an Axiom form so that users can export the contents of a Data Grid component to a spreadsheet file. This might be done as a substitute for printing the form, or to allow users to perform further manipulations of the data within a spreadsheet.

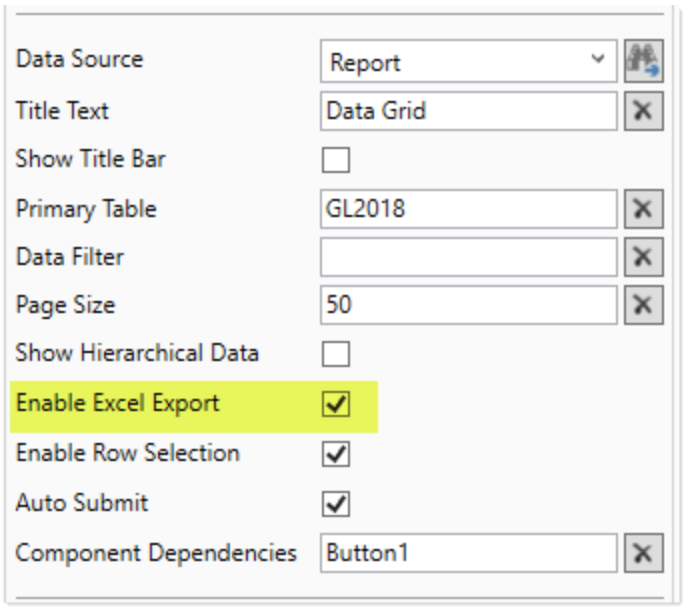
There are two options to export a data grid to a spreadsheet:

- Use the built-in option **Enable Excel Export** in the component properties.
- Use a separate Button component with the **Export Grid** command.

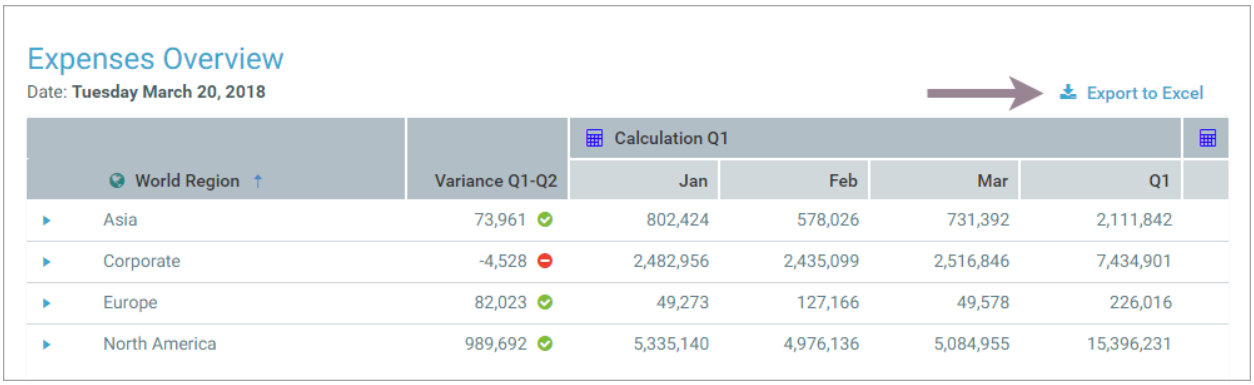
This topic applies to Data Grid components used in Axiom forms. If you want to export a Formatted Grid to a spreadsheet, see Exporting Formatted Grid contents to a spreadsheet.

Using the built-in export feature

To use the built-in export feature, enable the option **Enable Excel Export** in the Data Grid component properties.



When this option is enabled, an export button automatically displays at the top right corner of the component. Users can click this button to export the grid contents to a spreadsheet.



NOTE: If the grid columns do not fill the entire width of the component, the export button is still positioned at the top right of the component (not at the top right of the last column). In this case, you can either re-size the component so that it matches the total width of the columns, or you can use the Export Grid command with a separate Button component instead.

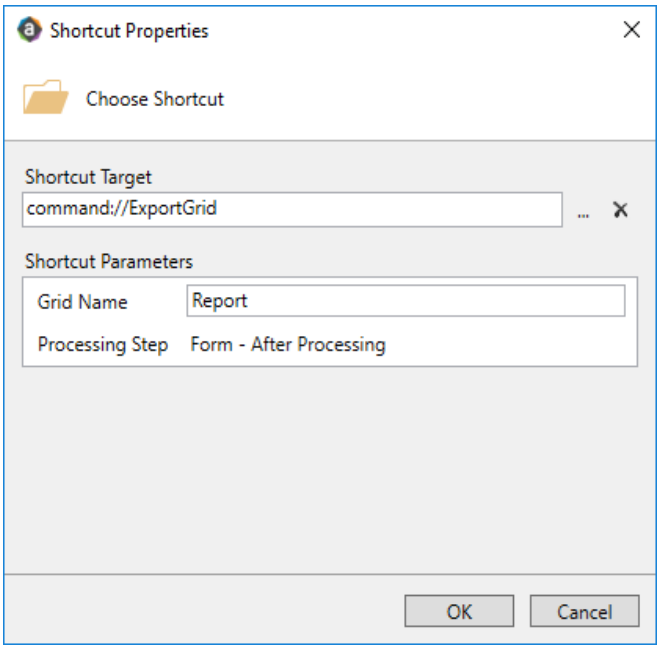
Using the Export Grid command

The **Export Grid** command can be used with a separate Button component as an alternative to the built-in export option. The primary reason to use the command is if you want to position the export button in a different location.

To start off, add the Button component to the Axiom form canvas and then configure the properties as desired. The button text should be defined as something like "Export Grid to Spreadsheet". You can then configure the **Command** for the component as follows:

1. In the Button component properties, click the **[...]** button to the right of the **Command** box.
2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.
3. In the **Axiom Explorer** dialog, navigate to the **Command Library**. Select the **Export Grid** command, then click **Open**.

   The Export Grid command is now listed as the shortcut target, and the relevant shortcut parameters are now available.
4. In the shortcut parameters, type the component name of the grid that you want to export, then click **OK** to close the Shortcut Properties dialog.



Example Shortcut Properties dialog

The button can now be used to export the contents of the specified grid.

Export behavior

When the grid data is exported, the behavior is as follows:

- The full contents of the data grid are exported (all pages). Column group headers and icons are omitted from the export. If the grid uses **Show Hierarchical Data**, this is disabled for the export and all hierarchical columns are exported as a flat list.

NOTE: If the Data Grid component uses a HierarchicalGrid data source, then only the contents of the top grouping level are exported. All other grouping levels are ignored and cannot be exported.

- Default number formats are applied to the exported data based on the underlying column data type (and for integer and numeric columns, the numeric type). The specific number format as set in the grid is not preserved.
- User changes to the grid, such as changing the sort order or filtering a column, are not preserved. However, refresh variables and other changes made in the actual form source file will carry through to the export. Basically, the exported data is the same as what displays in the grid before the user interacts with any grid-specific features.
- Unlike when exporting Formatted Grids, the **Is Excel Export** setting on the Form Control Sheet *cannot* be used to modify the grid before export. The export process does not trigger a form update and the grid is not refreshed.
- Once the grid contents have been exported to a spreadsheet file, that file is downloaded to the browser. The file name is the **Title Text** if defined in the component properties, otherwise an internally generated name is used. The browser prompts the user to save or open the file. The specific behavior of this download prompt depends on the browser used.
- The export is not supported for use on iPad or other tablets.

Formatted Grid component

Using the Formatted Grid component, you can display information in a grid format—such as reporting data, lists of information, or grid "worksheets" with various user inputs. The Formatted Grid component takes a designated section of the source spreadsheet and displays the contents of those cells in the Axiom form.

You can apply formatting to the grid contents, such as font size and colors, background colors, text alignment, and borders. The primary way to apply formatting is by tagging each row and column in the grid with style names. This formatting approach is known as a thematic grid. Axiom Software also supports using the formatting defined in the spreadsheet to format the grid contents, but this is a deprecated approach that is primarily supported for backward-compatibility only.

Defining a formatted grid is a two-part process that requires the following:

- Creation of a Grid data source in the spreadsheet to define the portion of the spreadsheet to display in the form.
    - If the grid uses thematic formatting, this process includes tagging rows and columns in the data source with the desired style for display in the form. The styles determine formatting such as fonts, colors, and borders. The formatting in the spreadsheet will be ignored, with the exception of number formats.

- If the grid uses spreadsheet formatting, this process includes formatting that portion of the spreadsheet so that it will display as you want within the form. Spreadsheet formatting such as font, color, and border settings will all be carried over into the form.

- Placement and configuration of a Formatted Grid component on the Axiom form canvas.

The formatted grid can be used for data display only, or to support rich interactivity such as input cells, combo boxes, and more. Formatted grids are very flexible and support many different features. This topic discusses how to create the data source and configure the component properties. For more information on using the other special features of formatted grids, see Using Grids.

Data source tags

A Formatted Grid component must have a defined data source within the file to indicate the data for the grid. The tags for the data source are as follows:

Primary tag

[Grid;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Formatted Grid component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Row]

Each row flagged with this tag (or the `[Fixed]` tag) defines a row of values to display in the grid.

[Fixed]

Optional. This tag can be used instead of the `[Row]` tag to flag a row as a non-scrolling header in the grid. Only the top row or rows in the grid can be fixed. If the `[Fixed]` tag is placed below a `[Row]` tag, the row will not be fixed and instead displays as a normal row.

The deprecated tag `[Header]` has the same effect. However, this legacy tag should not be used in new data sources.

If a `[RowID]` column is being used, fixed rows are not selectable.

[ColumnWidth]

Optional. Specifies a width for each column, in pixels or as a percent of the overall grid width. For more information, see Setting column sizes for Formatted Grids.

[ColumnStyle]

Thematic grids only. Specifies the column style for each column, to determine the formatting applied to each column. For more information about applying styles to thematic grids, see Using thematic formatting for Formatted Grids.

The data source can have one or more `[ColumnStyle]` rows. The first `[ColumnStyle]` row should be placed at the top of the data source, before any content rows (meaning rows tagged with `[Row]` or `[Fixed]`). The specified column style at the top of the data source will apply to the contents in that column until another `[ColumnStyle]` row is found. The column style in that row will then be used until another `[ColumnStyle]` row is found, and so on.

[DrillDownColumns]

Optional, thematic grids only. The presence of this tag enables data drilling for the grid, and this row is used to flag the columns to be included in the drill results. For more information on drilling tags, see Configuring the Grid data source for data drilling.

Column tags

[Column]

Each column flagged with this tag (or the `[Fixed]` tag) defines a column of values to display in the grid.

[Fixed]

Optional. This tag can be used instead of the `[Column]` tag to flag a column as a non-scrolling column in the grid. Only the leftmost column or columns can be fixed. If multiple columns are fixed, they must be continuous within the grid. If the `[Fixed]` tag is placed to the right of a `[Column]` tag, it will be ignored and treated as a normal grid column.

This tag is ignored if either of the following options are enabled in the component properties: **Fit Columns**, **Extended Height**. The columns will be treated as normal grid columns.

[RowHeight]

Optional, thematic grids only. Specifies a height for each row in pixels, or as a percent applied to the row style height. The row height is typically set by the row style, so this column is only intended for cases where you need to override the style or set the row to a height that is not supported by the available styles. For more information, see Setting row sizes for Formatted Grids.

[RowID]

Optional. Enables the ability to select rows in the grid. This column can contain any value that uniquely identifies each row in the grid, such as numbers or names. This is only necessary if you want to implement interactivity for the form based on the currently selected row of the grid. If you do not need this column, it can be omitted, and then users will be unable to select rows in the grid. For more information, see Selected row.

[RowStyle]

Thematic grids only. Specifies the row style for each row, to determine the formatting applied to each row. The row style applies to the entire row, except where it is overridden by a column style. For more information about applying styles to thematic grids, see Using thematic formatting for Formatted Grids.

[DrillDownRows]

Optional, thematic grids only. When enabling data drilling for the grid, this column is used to flag the rows that are eligible for drilling. For more information on drilling tags, see Configuring the Grid data source for data drilling.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows a Grid data source tagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or another data query method; this example uses fixed sample data in order to show the placement of the tags to the data. This example is for a thematic grid:

	B	C	D	E	F	G	H	I	J	K
2										
3		[Grid;NewTags]	[rowstyle]	[fixed]	[fixed]		[Column]	[Column]	[Column]	[Column]
4		[columnwidth]		10px	200px		80px-120px	80px-120px	80px-120px	80px-100px
5		[fixed]	row, bg-s2				[Format;Columns	Current Month Feb - 18		
6		[columnstyle]		col			colhead, right	colhead, right	colhead, right	colhead, right
7		[fixed]	row, bg-s2, row-height-18				2/14	2/14		
8		[fixed]	row, bg-s2, row-height-18, pad-b-4				Actual	Budget	Variance	Var %
9		[columnstyle]		col			col	col	col	col
10		[row]	row, bold, pad-t-4, row-height-18, pad-B-4		[Format;TargetCell=+1;]	Patient Revenue				
11		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Inpatient	123,456,789	123,456,789	123,456	5%
12		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Outpatient	123,456,789	123,456,789	123,456	5%
13		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Other Patient Rev	123,456,789	123,456,789	123,456	5%
14		[row]	row, bold, brd-tb-2		[Format;ColumnStyle=col, inc	Total Patient Reve	123,456,789	123,456,789	123,456	5%
15										
16		[row]	row, bold, pad-t-4, row-height-18, pad-B-4		[Format;TargetCell=+1;]	Patient Revenue				
17		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Inpatient	123,456,789	123,456,789	123,456	5%
18		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Outpatient	123,456,789	123,456,789	123,456	5%
19		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Other Patient Rev	123,456,789	123,456,789	123,456	5%
20		[row]	row, bold, brd-tb-2		[Format;ColumnStyle=col, inc	Total Patient Reve	123,456,789	123,456,789	123,456	5%

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Formatted Grid**. You can also highlight a range of data first and then use the

wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

When the form is rendered, this example grid displays as follows:

	Current Month Feb - 18			
	2/14 Actual	2/14 Budget	Variance	Var %
Patient Revenue				
Inpatient	123,456,789	123,456,789	123,456	5%
Outpatient	123,456,789	123,456,789	123,456	5%
Other Patient Revenue	123,456,789	123,456,789	123,456	5%
Total Patient Revenue	**123,456,789**	**123,456,789**	**123,456**	**5%**
Patient Revenue				
Inpatient	123,456,789	123,456,789	123,456	5%
Outpatient	123,456,789	123,456,789	123,456	5%
Other Patient Revenue	123,456,789	123,456,789	123,456	5%
Total Patient Revenue	**123,456,789**	**123,456,789**	**123,456**	**5%**

Notice the difference between the formatting in the form versus the formatting (or lack thereof) in the spreadsheet. The formatting in the spreadsheet is ignored when using a thematic grid. The formatting in the form is determined by the styles applied to the rows and columns in the data source. For example, the subtotal rows use a style such as **row,bold,brd-tb,brd-color-s6**, to apply bold text and borders to the row.

If desired, you could format the data in the spreadsheet, and that formatting would be ignored when the form is rendered. For example, you might want to bold the header rows and put underlines on the subtotal rows, just so the data is easier to read in the spreadsheet when form designers are working in the file.

This second example shows a data source for a spreadsheet-formatted grid. Notice that the thematic-only columns for the data source have been removed, and formatting has been applied to the spreadsheet.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
8															
9		[Grid;Income]			[Fixed]			[C [Column]		[C [Column]		[C [Column]		[[Column]	[Co
10		[Fixed]			**Budget to Actuals**										
11		[Fixed]			**Through Period 5**							**Variance**		**Variance**	
12		[Fixed]			**Account Category**			**Budget**		**Actual**		**Dollars**		**Pct.**	
13		[Row]													
14		[Row]			Revenue			68,315,310		69,631,147		1,315,837		1.93%	
15		[Row]			Cost of Goods Sold			18,290,732		21,607,003		3,316,271		18.13%	
16		[Row]			Margin			50,024,578		48,024,144		(2,000,434)		-4.00%	
17		[Row]													
18		[Row]			Payroll			20,893,926		23,527,142		2,633,216		12.60%	
19		[Row]			Travel			165,050		3,512,609		3,347,559		2028.21%	
20		[Row]			Other Expenses			2,660,620		3,884,317		1,223,697		45.99%	
21		[Row]			Overhead			7,542,849		8,502,979		960,130		12.73%	
22		[Row]			Marketing			97,867		411,084		313,217		320.04%	
23		[Row]			Taxes			525,220		600,948		75,728		14.42%	
24		[Row]			Other Income/Expense			50,100		472,915		422,815		843.94%	
25		[Row]			Total Expenses			31,935,632		40,911,995		8,976,363		28.11%	
26		[Row]													
27		[Row]			Net Income			18,088,946		7,112,149		(10,976,797)		-60.68%	
28		[Row]													

When the form is rendered, this example grid displays as follows:

Budget to Actuals

Through Period 5 **Account Category**	**Budget**	**Actual**	**Variance** **Dollars**	**Variance** **Pct.**
Revenue	68,315,310	69,631,147	1,315,837	1.93%
Cost of Goods Sold	18,290,732	21,607,003	3,316,271	18.13%
Margin	50,024,578	48,024,144	(2,000,434)	-4.00%
Payroll	20,893,926	23,527,142	2,633,216	12.60%
Travel	165,050	3,512,609	3,347,559	2028.21%
Other Expenses	2,660,620	3,884,317	1,223,697	45.99%
Overhead	7,542,849	8,502,979	960,130	12.73%
Marketing	97,867	411,084	313,217	320.04%
Taxes	525,220	600,948	75,728	14.42%
Other Income/Expense	50,100	472,915	422,815	843.94%
Total Expenses	31,935,632	40,911,995	8,976,363	28.11%
Net Income	18,088,946	7,112,149	(10,976,797)	-60.68%

Component properties

You can define the following properties for a Formatted Grid component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Grid Formatting	Specifies how the contents of the grid are formatted. Select one of the following: • **Thematic** (default): Grid contents are formatted based on the styles applied to each column and row in the data source. Spreadsheet formatting is ignored (except for number formats). For more information, see Using row and column styles in a thematic grid. • **Spreadsheet**: Grid contents are formatted based on the formatting defined in the spreadsheet. For more information, see Using spreadsheet formatting for a Formatted Grid. **IMPORTANT:** The spreadsheet formatting option primarily exists to support backward-compatibility for forms that were created prior to the introduction of the thematic grid (in version 2016.1). For any new grids, we strongly recommend using the thematic grid, as it will be the focus for all new enhancements moving forward.
Data Source	The data source for the grid. You must have defined the data source within the file using the appropriate tags in order to select it for the grid. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading. **NOTE:** Asynchronous loading cannot be used with embedded forms. If the Formatted Grid component will be used inside an embedded form, it must use Inline loading.

Item	Description
Selected Row ID	The currently selected row in the grid. This setting is optional and should only be used if you want users to be able to select a row in the grid to impact the Axiom form in some way. The data source must have a `[RowID]` tag in order to use this feature. This setting serves two purposes: • It defines the initially selected row in the grid, if you want the grid to start with a particular row selected. You can leave this blank to specify that no row is selected, or enter a row ID from the RowID column in the data source. • When a user views the form and selects a row in the grid, the row ID of the user's selection will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected row in the grid. For more information, see Selected row. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected row ID to be read from and written to the specified cell reference instead of directly within the Selected Row ID cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected row ID is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Auto Submit	Specifies whether the Axiom form automatically updates when a user changes the state of the grid. By default this is enabled, which means that the form automatically updates in response to user activity such as: • Changing the selected row (if using RowID) • Editing an unlocked cell in the grid • Interacting with a special formatted grid feature, such as a drop-down list or a check box If this setting is disabled, then edits made to the grid will not trigger an update. The changes will be submitted back to the source file when another component triggers an update, such as a Button component.

Item	Description
Use Lightweight Auto Submit	Specifies whether a special lightweight auto-submit behavior applies to input cells in the grid. If enabled, the following items are affected: simple unlocked cells and TextArea tags. This option is intended for cases where users are inputting numbers into the grid, and you want to update total formulas and other calculations that reference these inputs, but without submitting all grid values or triggering a full form update. **NOTE:** This option only applies to thematic grids. When enabled, the following behavior applies: • When a user edits an unlocked cell or a text box, only that new value is submitted back to the form source document on the server. • The source document is calculated so that any formulas referencing the changed cell are updated. No data refresh occurs, just a calculation. • If any of the following cell types in the grid have modified values after the calculation, the form web page is updated to display these new values: regular locked or unlocked cells, cells with Format tags, and cells with TextArea tags. No other form values are submitted, and no other form components are updated. The regular form update process does not occur. This option can be enabled by itself or in conjunction with **Auto Submit**. This option overrides auto-submit behavior for the affected cells, but all other cells can continue to use the regular auto-submit behavior or not as desired.
Save On Submit	Specifies whether a save-to-database occurs when a form update is triggered by this component. • If disabled (default), then changing this component does not trigger a save-to-database. • If enabled, then a save-to-database will occur as part of the form update process when this component triggers an update. The save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. A save-to-database process must be enabled and configured within the source file. For more information, see Saving data from an Axiom form. This setting only applies if Auto Submit is enabled for the component. If you are not using the auto-submit behavior but you do want to save data to the database from the Axiom form, then you should instead enable Save on Submit for the Button component that you are using to trigger the update process.

Item	Description
Title Text	The title text for the component. This text displays in the title bar for the component within the Axiom form, if the title bar is enabled. If the title bar is disabled, then this text does not display at all in the form. It is assumed that if you are not using the title bar, then you have defined title text in the grid itself. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Fit Columns	Specifies whether columns are scaled to fit within the component width. This option only applies to spreadsheet-formatted grids. • If enabled (default), then columns in the data source will be proportionally scaled (larger or smaller) to fit the width of the grid component on the form canvas, based on the column width in the sheet. • If disabled, then columns will not be scaled smaller to fit the width of the grid component. Instead, if the total column width exceeds the width of the grid component, then the columns will maintain their width as defined in the sheet and the grid will scroll horizontally. However if the total column width is less than the width of the grid component, the columns will still be scaled larger to fit the width but in this case the extra width is allotted equally instead of proportionally. If this option is enabled for a spreadsheet-formatted grid, then the `[ColumnWidth]` tag in the data source cannot be used to control column widths. If you want to use this tag instead of using the spreadsheet column widths, then you must disable Fit Columns. Thematic grids do not support "fit column" behavior for the component, though column sizes can be specified so that columns always fit within the component (for example, by using percentage-based sizing). For more information, see Setting column sizes for Formatted Grids.

Item	Description
Collapse Height	Specifies whether the component automatically collapses in height if the configured component height is greater than the content to be shown in the component. You should leave this option disabled if you want the component height to always be the same, no matter how much content is available to display. If the content exceeds the component height, the component will have a vertical scroll bar. If the content does not fill the component, then there will be blank space between the final row of content and the bottom edge of the component. If you enable this option, then when the content does not fill the component, the component will auto-shrink to fit the content instead of maintaining its configured height. The behavior if the content exceeds the component height is the same (vertical scroll bar).
Extended Height	Specifies whether the height of the grid should extend to show all rows of the data source. By default this is disabled. If the rows included in the data source exceed the height of the grid component on the form canvas, then a vertical scroll bar will be present on the grid, allowing users to scroll to view all rows. If this option is enabled, then the height of the grid will automatically extend downward to include all rows. If the rows exceed the size of the form window, then the user can scroll the window to view all rows. Use of this option limits certain features that can be used in the grid and for the form. It is only intended to be used for special situations that require a non-scrolling grid. For more information, see Setting row sizes for Formatted Grids.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

For Formatted Grid components, the grid-level style only impacts the external grid container; it does not affect the internal grid contents.

- If the grid is a thematic grid, then row and column styles should be applied to the data source to define the formatting for the grid contents. For more information, see Using row and column styles in a thematic grid.

- If the grid is a spreadsheet grid, then the grid contents inherit the formatting defined in the spreadsheet. For more information, see Using spreadsheet formatting for a Formatted Grid.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Using the Data Source Assistant with a Grid data source

You can use the Data Source Assistant to:

- Add column and row tags to the data source.
- Apply styles to rows and columns. For information on using the Data Source Assistant to apply styles, see Using row and column styles in a thematic grid.
- Edit content tags. For information on using the Data Source assistant to work with content tags, see Creating and editing content tags in Formatted Grids.

The **Selection Editor** section of the Data Source Assistant is blank if you have your cursor in any of the following areas:

- In the `[ColumnWidth]` row or the [RowHeight] column.
- In the `[RowID]` row.
- In a cell that contains a formula, or plain text or numbers.

You must populate these areas manually as needed.

Interactivity options for Formatted Grids

Formatted Grid components support interactivity in the following ways:

- The selected row in the grid can be submitted back to the source file to impact another component, such as to change the data displayed in a chart based on the currently selected row.
- Individual cells in the grid can be flagged as editable, so that the changed contents of the cell are submitted back to the source file. This can support basic user inputs.
- Special content tags are available to present certain interactive controls within individual cells, such as drop-down lists and check boxes. These items will either submit changed contents back to the source file or perform an action, depending on the specific tags used.

For general information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Selected row

If the `[RowID]` column is used in the data source, then users viewing the Axiom form can select a row in the grid. The selected row becomes highlighted in the form, and the selected row is submitted back to

the source file, using the value in the `[RowID]` column. The selected row ID is written to the **Selected Row ID** setting on the Form Control Sheet.

If you want the Axiom form to respond to the currently selected row, then you must set up the file so that another component references the selected row ID and changes based on it.

For example, an Axiom form could contain a chart that dynamically changes data based upon the currently selected row in a grid. The row IDs in the grid could contain region names, like North, South, etc. When a user selects the row with an ID of North, that selection is written back to the source file.

The chart could then reference that value to filter the data in the file, or to show the appropriate series for the selected value. So as the user selects rows in the grid, the chart updates to show data relating to the selected row.

Note the following when setting up the row IDs:

- Row IDs must be unique. Only one row in the grid can be selected. If IDs are not unique, the grid will not behave as expected.
- If a row does not have a row ID (the cell is blank), then that row is not selectable. Clicking on that row will not result in a row ID being written back to the source file.
- Rows tagged with `[Fixed]` are not selectable, regardless of whether the row has a row ID. Clicking on a fixed row will not result in a row ID being written back to the source file.

The appearance of the hover row and the selected row is determined by the style or skin (in that order). For spreadsheet-formatted grids, only the skin applies. For thematic grids, the style that impacts the formatting is the row style, not the component-level style.

Editable cells

If any cells in the grid are unlocked, then users viewing the Axiom form can edit the contents of that cell. The changed contents are submitted back to the source file, to the corresponding cell in the data source on the sheet.

The determiner of whether a cell can be edited is the cell's locked status (**Locked** must be unchecked in the cell properties). The unlocked cell will be displayed in the grid as follows:

- In thematic grids, the editable cell renders as a text box. If the cell uses a numeric format, then it will be treated as a numeric text box without a defined range. Otherwise, the cell is treated as a regular text box.

  In some cases you may want the editable cell to display as a normal cell by default, and only render as an editable text box when the user clicks into it. To achieve this effect, you can use a special column style named `click-to-edit`. This style requires the form to use the Web Client Container.

- In spreadsheet-formatted grids, as an editable cell. No special formatting is applied to indicate that the cell is editable. You should format the spreadsheet as desired—for example, you may want to display the cell with a border and a background color as a signal to the user that they can edit the cell.

In most cases this option would be used to save the user's inputs to the database (although it can also be used to simply impact the contents of the form in some way). If saving edited data to the database, remember that the file itself is not saved. For more information, see Saving data from an Axiom form.

Alternatively, the TextArea content tag can be used to present editable cells to users in an Axiom form. The content tag provides greater control over the display and behavior of the editable cell. For more information, see Using text boxes in Formatted Grids.

Content tags

Formatted grids support a series of content tags that can be used to format cell content and apply special features to a cell. Content tags can be used to present user inputs within a grid—such as combo boxes, text boxes, and check boxes—as well as to display symbols or hyperlinks. These tags have no effect in the source file, but when the Axiom form is rendered, the cell will display according to the content tag configuration.

The content tags can be used in any cell that falls within the data source for the formatted grid. The tags use the following basic syntax:

`[FormatTag;Parameter1=Value;Parameter2=Value]`

Where the *FormatTag* is the tag that defines what you want to display in the cell, with one or more parameters specific to that tag separated by semicolons.

For more information, see Using content tags in Formatted Grids.

Design considerations

One of the most important design considerations when working with interactive formatted grids is whether the data in your grid is being sourced using an Axiom query.

For example, imagine that you have set up an editable cell in a grid that is being populated by an Axiom query. When a user edits the cell in the Axiom form, the change is submitted back to the source file (either by the grid being set to Auto-Submit or by using a separate Button component). However, after the changed values are submitted, the source file is refreshed, including running Axiom queries. If the query that populates the formatted grid is run at this time, then it will simply overwrite the edit made by the end user. This means you must carefully think about when each Axiom query should be run, and configure the refresh options and/or active status of the query to achieve the desired results.

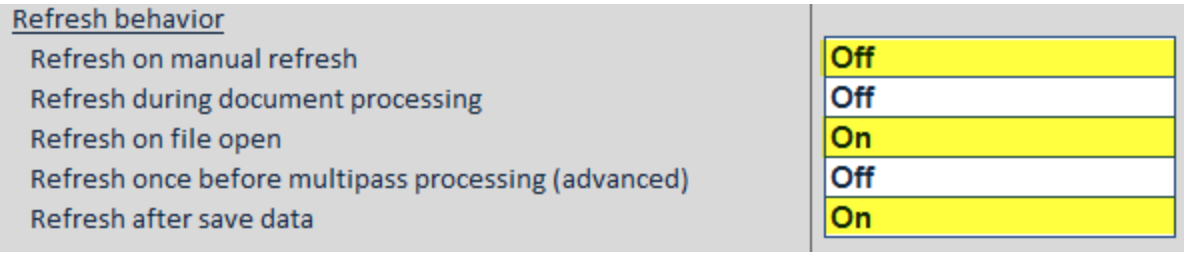
You may want to use the **Triggering Component** setting on the Form Control Sheet to control when an Axiom query should be run. When an update is triggered for an Axiom form, the name of the component that triggered the update is written to this field. For example, if a formatted grid is set to Auto-Update and a user edits a cell in that grid, then when the form is updated as a result of this change the triggering

component is set to the name of the formatted grid. You can set up a formula in the Axiom query settings so that the query is inactive when the triggering component is that formatted grid, and therefore the query will not run when the update occurs.

Alternatively you may be saving the user's inputs to the database, and therefore you want to run the query only in the following circumstances:

- When the form is first opened by the user, to initially populate the formatted grid.
- After data is saved to the database, to repopulate the formatted grid to include the recently saved data.

You can achieve this behavior by setting the refresh options for the Axiom query as follows:



You could also use a combination of these options—for example you might toggle the **Refresh on manual refresh** option on or off depending on whether the update was triggered by the formatted grid or not.

Formatting options for Formatted Grids

There are two different options to define the formatting for the rows and columns displayed in the grid:

- **Thematic formatting**: When using this option, you use special data source tags to specify styles to apply to each row and column of the grid. When the form is rendered, the grid cells display according to the applied styles. Formatting such as fonts, colors, borders, etc. are all determined by the applied styles instead of by the formatting in the spreadsheet. This option separates the formatting from the spreadsheet data, so that the entire look and feel of the grid can be changed simply by changing the styles.
    - If your form uses the default Axiom2018 skin, there is one set of row and column styles to cover the full range of grid formatting options. These styles combine together to directly specify the formatting to apply to the row or column. For example, the style may directly specify the font color and size, the borders and margins, and other formatting properties. Themes are not used with this new skin.

- If your form uses a legacy skin, the styles available and their formatting depend on the theme applied to the grid (either inherited from the form-level theme or set at the component level). These styles were designed to be semantic, such as a "total" row style to apply a predefined set of formatting to a total row. This approach has been deprecated, due to user feedback that requested the ability to apply more granular formatting. The legacy skins, themes, and styles can continue to be used, but they will not be the focus of enhancements going forward.

- **Spreadsheet formatting**: When using this option, you use spreadsheet features to format the cells in the data source as you want them to display in the form, including fonts, colors, borders, etc. When the form is rendered, the grid cells will display in basically the same way they do in the spreadsheet, with some limitations. The Formatted Grid component serves as a means to display a portion of the spreadsheet within the form.

IMPORTANT: The spreadsheet formatting option primarily exists to support backward-compatibility for forms that were created prior to the introduction of the thematic grid (in version 2016.1). For any new grids, we strongly recommend using the thematic grid, as it will be the focus for all new enhancements moving forward.

Setting column sizes for Formatted Grids

There are two ways to set column sizes for the data shown in Formatted Grid components:

- You can explicitly set column widths using the `[ColumnWidth]` tag in the data source. This applies to both spreadsheet-formatted grids and thematic grids.

  OR

- You can use the column width in the spreadsheet to determine the column sizing in the form. This only applies to spreadsheet-formatted grids.

Using the [ColumnWidth] tag

Using the `[ColumnWidth]` tag, you can set a specific size for each column in the Grid data source, using either pixels or percentages as the size units. This tag can be used with both thematic grids and spreadsheet-formatted grids.

To use the `[ColumnWidth]` tag:

- Add the tag to the control column of the Grid data source. Typically, the tag is placed near the top of the data source, above any content rows (meaning rows marked with `[Row]` or `[Fixed]`).

- If the grid is a spreadsheet-formatted grid, disable **Fit Columns**. If Fit Columns is enabled, then the `[ColumnWidth]` tag is ignored and instead the Fit Columns behavior applies (see the following section). This requirement does not apply to thematic grids as the Fit Columns option is always ignored for thematic grids.

Within the `[ColumnWidth]` row of the data source, enter a size for each content column that will be displayed in the form. This means any column that is flagged with `[Column]` or `[Fixed]`. Sizes can be entered as follows:

Size	Description	Examples
Specific size	Enter a single number to set the size of the column to that number. The size can be expressed in pixels or as percent.	50px 25%
Size range	Enter a minimum and maximum number to set the allowed size range for the column. The column will be sized somewhere in this range, depending on the other column sizes and the overall width of the component. Use a hyphen to separate the minimum and maximum number. The column will display at the minimum size by default. If the total of all column widths is less than the overall width of the component, then the column will "flex" wider, up to the maximum size. When mixing pixels and percents in a range, it is possible for the maximum number to end up smaller than the minimum number once the percent is resolved to pixels. In this case, the maximum number is effectively ignored and the column will not expand past its minimum number.	50px-100px 25%-40% 50px-25%
No size	If the column has no defined size, then the column will be sized as if it has a minimum size of 0px and an unlimited maximum size. Therefore its rendered size will depend on the other column sizes and the overall width of the component. The column will display at the minimum size by default. If the total of all column widths is less than the overall width of the component, then the column will "flex" wider, with no maximum size limit.	N/A

NOTE: If units are not specified on the number (px or %), then pixels are assumed. It is recommended to place units on all numbers to avoid confusion and to prevent Excel from auto-converting certain entries into dates.

If you are using percentages, the column width is determined using the total rendered width of the Formatted Grid component. For example, if you specify 25% for a column and the component width is 400 pixels, then the column will be 100 pixels wide. The component width may be a fixed size (such as

400px) or a relative size (such as 50% of the form or of the parent panel). If the component width is relative, then when the form is rendered the actual component width is calculated first and then the column width is calculated.

If the total of all column widths (using the fixed or minimum size) exceeds the overall width of the component, then a horizontal scroll bar will be present on the grid. In this case, you must define a width for all columns because any columns without a defined width are sized using the 0px minimum and therefore will not be visible.

If the total of all column widths is less than the overall width of the component, then the columns will be sized as follows:

- Any columns with a defined size range or with no defined size will be expanded wider. The excess width will be allocated among these "flex" columns evenly, except that no column with a defined range will exceed its defined maximum size. Any leftover width (for example, due to rounding) is given to the last column that has not exceeded its maximum size.
- If all columns have a defined size or range, and after expanding to the maximum range there is still excess width, then that width will display as a gap between the right edge of the last column and the right edge of the grid component.

If all columns use percent sizes, then the following applies:

- If the total of all column widths equals 100%, then the data source contents will exactly fit the component width.
- If the total of all column widths is less than 100%, then a gap will display between the right edge of the last column and the right edge of the grid component.
- If the total of all column widths is greater than 100%, then a horizontal scroll bar will be present on the grid.

It is possible to mix percents and pixels for different columns within the grid, and even within a single range. Each column will be sized according to the rules described above to determine the total column widths and therefore the overall grid display.

If the `[ColumnWidth]` tag is not present in the data source, then the grid is treated as follows, depending on the format type:

- **Thematic grid**: The overall width of the grid is allocated equally to all columns. The column width in the spreadsheet is ignored.
- **Spreadsheet-formatted grid**: The column width in the spreadsheet is used to determine the column sizing. See the following section for more information.

Using the column width in the spreadsheet

This option only applies to spreadsheet-formatted grids. Thematic grids do not use the column width in the spreadsheet.

To use the column width in the spreadsheet, remove the `[ColumnWidth]` tag from the data source (if it is present) and then size each column in the spreadsheet as desired. Columns will be sized as follows in the form, depending on whether the Fit Columns property is enabled in the component properties.

- If Fit Columns is enabled, then the columns will be auto-scaled to fit the width of the Formatted Grid component on the form canvas. If the overall width of the columns is less than the component width, the columns will be scaled wider. If the overall width of the columns is greater than the component width, the columns will be scaled narrower. The scaling is proportional based on the width of the columns as defined in the spreadsheet.
- If Fit Columns is disabled, then the columns will display in the grid based on their width in the spreadsheet. This is primarily intended for situations where the overall width of the columns exceeds the width of the component, and instead of shrinking column width you want the user to be able to scroll horizontally to see additional columns. Keep in mind the following:

  NOTE: If the overall width of the columns is *less* than the width of the component, then the columns will still be scaled to fit the width of the component. However in this case, the "extra" width is allotted to all columns evenly instead of proportionally. In other words, the initial column width is set based on the width in the spreadsheet, and then the extra width is divided by the number of columns and added to each column.

By default, Fit Columns is enabled for all new Formatted Grid components.

Printing considerations

If you intend to allow users to print the form by generating a PDF, then the overall column width should not exceed the component width. In other words, all columns must be visible in order to be printed.

The easiest way to achieve this is to configure the grid so that the columns never scroll. However, you may have a situation where you want the grid to be scrollable online but fitted to the component width when generating a PDF. You can do this by dynamically changing the grid configuration for the PDF—for example, to dynamically enable Fit Columns for a spreadsheet-formatted grid. For more information, see Configuring an Axiom form for printing.

Setting row sizes for Formatted Grids

The way that rows are sized in Formatted Grid components depends on the format type of the grid:

- **Thematic grids**: Rows are sized according to the row style applied to each row and the `[RowHeight]` column in the data source. The row height set in the spreadsheet is ignored.
- **Spreadsheet-formatted grids**: Rows are sized based on the row height in the spreadsheet.

In both cases, the overall number of rows that can be displayed in the grid (without scrolling) depends on the height of the Formatted Grid component on the form canvas. The grid will behave as follows depending on whether the rows in the data source can fit into the height of the component:

- If the overall height of all rows in the data source exceeds the height of the component, then a vertical scroll bar will be present on the grid. The user can use the scroll bar to view all rows.
- If the overall height of all rows in the data source takes up less space than the height of the component, then the grid behavior depends on the **Collapse Height** setting in the component properties. If disabled, then the grid remains at its configured height, and there will be blank space between the last row and the bottom edge of the component. If enabled, then the grid auto-shrinks vertically to fit the height of the row contents.

Alternatively, the **Extended Height** property for the Formatted Grid component can be used to automatically extend the height of the grid downward to accommodate the number of rows in the data source.

Setting row heights for thematic grids

Row heights in thematic grids are set as follows:

- Each row uses the height set by the row style by default. All base row styles include a specified row height (including the default row style that is used when no base style is specified). Additionally, you can apply add-on styles to set the row height, such as **row-height-18**. For more information on using row styles, see Using row and column styles in a thematic grid.
- If a row height is set in the `[RowHeight]` column of the Grid data source, this row height overrides the style. So if the style sets a row height of 12 pixels, but you enter 30 pixels into the `[RowHeight]` column, then the row height for that row is 30 pixels.

  Generally speaking, this column is only intended for cases where you need to override the row style or set the row to a height that is not supported by the available styles. It is not necessary to enter a row height for every row.

  NOTE: If the row style includes a minimum row height, this minimum also applies to the height set in the `[RowHeight]` column. For example, styles that impact the font size may include a minimum row height, to ensure that the row is tall enough to show the text.

The following entries are valid in the `[RowHeight]` column:

- **Value in pixels**: For example, you can enter `30` or `30px` to set the height to 30 pixels.
- **Value in percent**: Percent values are applied to the height defined by the row style. For example, if the row style is 12 pixels, you can enter 200% to size the row twice the height of the row style (24 pixels).
- **Auto**: You can enter the keyword `auto` to indicate that the row uses the height defined in the row style. This is for cases where you want to populate all cells of the `[RowHeight]` column for clarity, but you want some rows to use the height defined in the style.
- **Blank**: The row uses the height defined in the row style.

Using Extended Height to dynamically expand the grid

If Extended Height is enabled for a grid, then the grid will dynamically extend downward to display all rows in the data source. If the height of the grid exceeds the current window, then the user can scroll the window to view the rest of the grid. This option can be used with thematic grids and spreadsheet-formatted grids.

Use of Extended Height introduces some limitations to the Formatted Grid component. The option is only intended to be used for special situations that require non-scrolling grids with many rows, such as when the form needs to be printed.

- If Extended Height is enabled, no other components can be placed below the formatted grid on the form canvas. The extended grid will *not* "push" the other components down; instead the other components will continue to display at their fixed location on the canvas and the formatted grid will extend underneath the other components.

  There is one exception to this behavior. If a thematic grid with extended height is placed within a flow panel, the placement of the other components in the flow panel will adjust for the extended height. However, note that the height of the panel itself does not adjust (even if set to dock), so the **Overflow** property of the panel cannot be set to **Hidden**.

- If the grid is a spreadsheet-formatted grid, certain grid features are not compatible with use of the Extended Height option. This includes the following content tags: Checkbox, TextArea, and Sparkline. Also, disabling Auto-Submit will not work as expected if Extended Height is enabled. These limitations do not apply to thematic grids.

- Fixed rows and columns are not honored when Extended Height is enabled.

- Depending on the number of rows in the grid and the browser used to view the form, performance may be slow when viewing an extended grid online. If the Extended Height option is being used to facilitate printing, it is recommended to dynamically turn it on and off as needed (as discussed below) to preserve online performance of the grid.

- The form property **Scale to Fit** must be disabled in order for the grid to extend when viewing online. If Scale to Fit is enabled, then the Extended Height option is ignored and the grid will scroll as normal. For information on Scale to Fit, see Defining the canvas size of an Axiom form.

Printing considerations

If you intend to allow users to print the form by generating a PDF, then the overall height of all rows should not exceed the component height. In other words, all rows must be visible in order to be printed.

If you know that the rows will never exceed the component height, then no special treatment is necessary for printing. However, if a grid could have many rows, then you may want the grid to scroll online but use Extended Height for printing only. You can dynamically enable or disable Extended Height based on whether the form is currently being converted to a PDF. When a user creates a PDF of an Axiom form, the **Is PDF** setting on the Control Sheet is changed to **On**, and then the form is refreshed before the PDF is generated. The Extended Height setting could use a formula such as the following:

```
=IF(Control_Form!D22="On","On","Off")
```

In this example, Control_Form!D22 is the location of the **Is PDF** setting. Extended Height is disabled when users view the form online (Is PDF is Off). But when the form is converted to a PDF (Is PDF is On), then Extended Height is enabled so that all rows display in the PDF document.

NOTES:

- If a grid using extended height is placed within a panel, and the height of the extended grid exceeds the panel height, then the panel **Overflow** must be set to **Visible** in order to print the entire grid.
- When generating a PDF of a form for printing, the Scale to Fit setting is automatically disabled.

Using thematic formatting for Formatted Grids

By default, Formatted Grid components are set up to use thematic formatting—meaning, the **Grid Formatting** option for a Formatted Grid component is set to **Thematic**. We strongly recommend creating all new grids as thematic grids, as thematic grids support more options than spreadsheet-formatted grids, and will be the focus of all new grid enhancements moving forward.

From a formatting perspective, thematic grids have many advantages over spreadsheet-formatted grids. By using styles to define formatting instead of using the spreadsheet formatting, it is much easier to apply consistent formatting to many different grids in many different forms. It is also much easier to update the formatting—if a change is made to the style then that change automatically flows through to all grids using the style. When using spreadsheet formatting, you would have to manually edit each grid for the desired change.

When using a thematic grid, the grid formatting is determined by the following elements:

- **The style used by the Formatted Grid component.** This style is set in the component properties, and only affects the grid "container-level" formatting, such as an external border. It does not affect the formatting of the grid contents.
- **The row and column styles defined in the Grid data source.** Each row and column can be flagged with different styles to set the formatting as appropriate for its contents. Styles can apply formats like bold font, borders, background colors, and more.

NOTE: If your form uses a legacy skin (any skin other than the default Axiom2018 skin), then the grid formatting is also affected by the assigned theme. By default, this is inherited from the form-level theme, though it can be set at the component level if necessary. Different themes have different default formatting, and provide access to different row and column styles. For example, the form theme may be set to Report to support display of data, or to Worksheet to support user inputs. Themes do not apply when using the Axiom2018 skin, because that skin uses a different set of styles that are designed to accommodate all of the display variations that used to require themes.

Using row and column styles in a thematic grid

When using a thematic Formatted Grid component, the contents of the grid are formatted by specifying row and column styles within the Grid data source. The formatting defined in the spreadsheet is mostly ignored, with a few exceptions.

IMPORTANT: This topic focuses on applying row and column styles when the form uses the default Axiom2018 skin. If you are using a legacy skin, the available styles are not the same and work slightly differently. For more information on the legacy approach, see Using theme-based row and column styles (legacy skins).

Formatting inherited from spreadsheet

When using a thematic grid, all spreadsheet formatting is ignored except for the following:

- Number formats are inherited from the spreadsheet. If you want numbers to display using decimals, percentage signs, comma separators, etc., you must format the cells as appropriate in the spreadsheet.
- If a spreadsheet cell uses General alignment, the grid will honor the spreadsheet alignment behavior unless alignment is explicitly defined in the column style. By default, the General alignment is left-aligned for text and right-aligned for numbers. For example, if a column uses the default **col** style with no alignment specified, then text and numbers will be auto-aligned as they would in the spreadsheet. But if a column instead uses styles **col, right**, then all column contents will be right-aligned, because the **right** style explicitly adds right text alignment.
- If a cell is unlocked in the spreadsheet, that cell is editable in the grid and will display as an editable text box. If you do not want the cell to be editable, you must lock the cell. All cells start off locked by default.

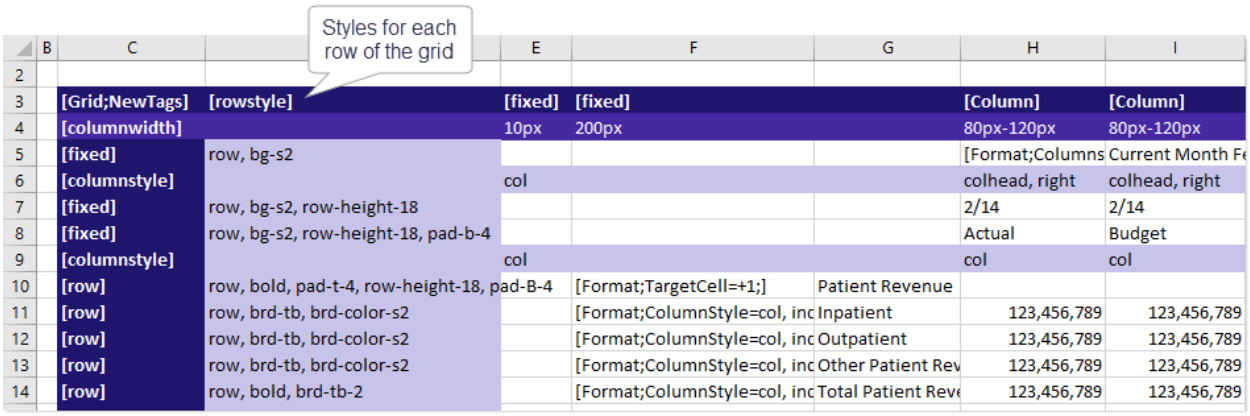
Using row styles

The row style applies formatting to the entire row, except where the formatting is overridden by a column style.

Row styles are placed in the `[RowStyle]` column, which must be present in the control row of the Grid data source. This tag is included by default when you use **Create Axiom Form Data Source > Grid** to create the data source. If the tag is not present, you can manually add it to the data source, or you can use the Data Source Assistant to add it. The tag can be placed anywhere to the right of the primary `[Grid]` tag, but for ease of viewing the data source it is recommended to place it before any content columns.

For each row in the grid that is tagged with either `[Fixed]` or `[Row]`, you can enter one or more style names in the `[RowStyle]` column.

- The style names for a row can include one base style plus any number of add-on styles, separated by commas. For example, you can specify just **row** to use the default row formatting, or you can include add-on styles such as **row,brd-tb,brd-color-s6,bold**. In this example the add-on styles add borders and bold text to the row. You can also specify just the add-on styles, and the **row** base style will be assumed. See Using base styles and add-on styles.
- The specified styles apply to that row only, for the entire row. If a column style defines a formatting property that conflicts with the row style, the property from the column style will be used in that column only. For example, the row style may specify normal weight font, but a particular column may be styled to use bold font. In that case, the row cell in that particular column will be bold.
- If no style is defined for a row (or if no `[RowStyle]` column is present), the default row formatting is used. This is equivalent to using the **row** style without any add-on styles. Although you can leave rows blank in order to use the **row** style, it is recommended to flag all rows with a style so that form designers reviewing the source file can see exactly how the grid is meant to be formatted.
- You can type style names manually, or you can use the Data Source Assistant to see the available style names and populate the current cell. It is strongly recommended to use the Data Source Assistant to apply styles, both to ensure that you are using correct style names and syntax, and because the Data Source Assistant provides an easy way to view all style options. For more information on using the Data Source Assistant, see Using the Data Source Assistant to apply row and column styles.
- If the grid data is being populated by a rebuild or insert Axiom query, remember that any rows coming from the Axiom query must have a row tag and row styles defined in the calc method, within the relevant columns. When the Axiom query is run and rows are dynamically inserted into the sheet, they will be brought in with the necessary tags for display and formatting within the grid. Any header and total rows that are outside of the Axiom query data range (and therefore static) can be tagged directly on the row.
- For a full list of available row styles, see Thematic grid style reference.



	B	C		E	F	G	H	I
2								
3		[Grid;NewTags]	[rowstyle]	[fixed]	[fixed]		[Column]	[Column]
4		[columnwidth]		10px	200px		80px-120px	80px-120px
5		[fixed]	row, bg-s2				[Format;Columns	Current Month Fe
6		[columnstyle]		col			colhead, right	colhead, right
7		[fixed]	row, bg-s2, row-height-18				2/14	2/14
8		[fixed]	row, bg-s2, row-height-18, pad-b-4				Actual	Budget
9		[columnstyle]		col			col	col
10		[row]	row, bold, pad-t-4, row-height-18, pad-B-4		[Format;TargetCell=+1;]	Patient Revenue		
11		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Inpatient	123,456,789	123,456,789
12		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Outpatient	123,456,789	123,456,789
13		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Other Patient Rev	123,456,789	123,456,789
14		[row]	row, bold, brd-tb-2		[Format;ColumnStyle=col, inc	Total Patient Reve	123,456,789	123,456,789

Example [RowStyle] column in a Grid data source

Using column styles

The column style sets the formatting for the column. Unlike row styles, column styles can be changed at any point in the column. You can start the column with one style, and then change it to a different style four rows later, and so on. You can also specify column styles on a per cell basis, to be applied to that cell only.

Column styles are placed in the `[ColumnStyle]` row. The Grid data source must have at least one `[ColumnStyle]` row in the control column of the data source, placed at the top of the data source (meaning before any content rows). This initial `[ColumnStyle]` row defines the starting style for each column. This style will continue to be applied to the column until another `[ColumnStyle]` row is encountered, at which point that style will be applied until another `[ColumnStyle]` row is encountered, and so on.

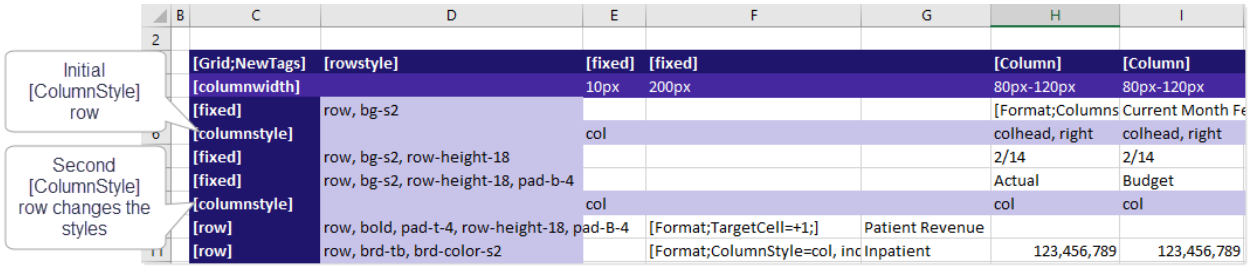
The initial `[ColumnStyle]` tag is included by default when you use **Create Axiom Form Data Source > Grid** to create the data source. If the tag is not present, you can manually add it to the data source, or you can use the Data Source Assistant to add it.

For each column in the grid that is tagged with either `[Fixed]` or `[Column]`, you can enter one or more style names in the `[ColumnStyle]` row(s).

- The style names for a column can include one base style plus any number of add-on styles, separated by commas. For example, you can specify just **col** to use the default column formatting, or you can include add-on styles such as **col,bg-p3,bold**. In this example the add-on styles add a background color and bold text to the column. You can also specify just the add-on styles, and the **col** base style will be assumed. See also Using base styles and add-on styles.
- If no style is defined for a column (or if no `[ColumnStyle]` row is present), the default column formatting is used. This is equivalent to using the **col** style. Although you can leave columns blank in order to use the **col** style, it is recommended to flag all columns with a style so that form designers reviewing the source file can see exactly how the grid is meant to be formatted.

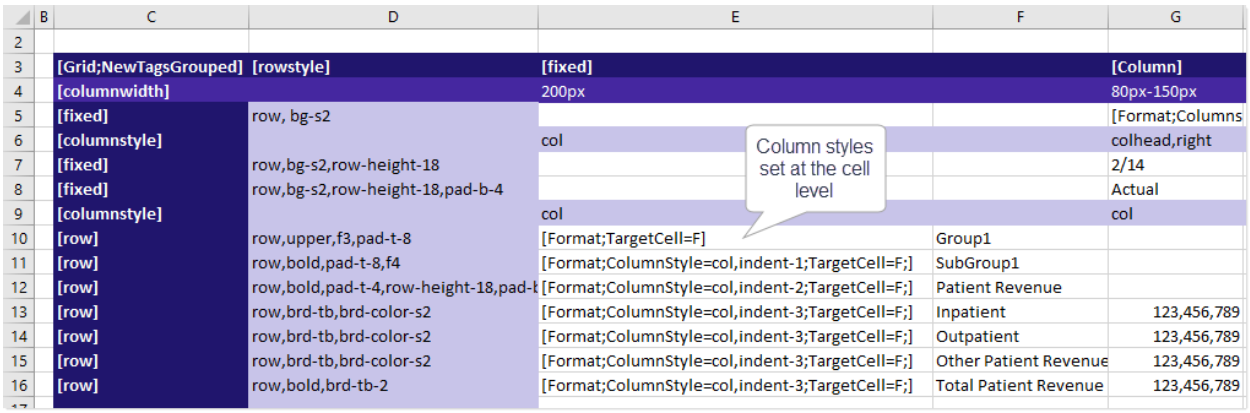
- When using multiple `[ColumnStyle]` rows, if a column is left blank then the previously specified style will continue to apply. Blank will not be interpreted as no style in this case (assuming there is a previously specified style in a prior `[ColumnStyle]` row).
- The column style can be overridden at the cell level when using content tags such as Format, TextArea, or Symbol. This can be used to format different levels of header text in a label column, or to apply conditional formatting to cells (such as to flag problem values in red). If a column style is specified in a content tag, the current column style from the `[ColumnStyle]` row is ignored, and the style specified in the tag is used for that cell only.
- You can type style names manually, or you can use the Data Source Assistant to see the available style names and populate the current cell. It is strongly recommended to use the Data Source Assistant to apply styles, both to ensure that you are using correct style names and syntax, and because the Data Source Assistant provides an easy way to view all style options. For more information on using the Data Source Assistant, see Using the Data Source Assistant to apply row and column styles.
- For a full list of available column styles, see Thematic grid style reference.

The following example screenshot shows a thematic grid with two `[ColumnStyle]` rows. In this example, columns H and I start out using the **colhead** style, but then later change to the **col** style. This is to display column header shading and bold in the first few rows, but then revert to "normal" column formatting afterward. You can have as many `[ColumnStyle]` rows in the grid as needed.



	B	C	D	E	F	G	H	I
2								
		[Grid;NewTags]	[rowstyle]	[fixed]	[fixed]		[Column]	[Column]
		[columnwidth]		10px	200px		80px-120px	80px-120px
		[fixed]	row, bg-s2				[Format;Columns	Current Month Fe
6		[columnstyle]		col			colhead, right	colhead, right
		[fixed]	row, bg-s2, row-height-18				2/14	2/14
		[fixed]	row, bg-s2, row-height-18, pad-b-4				Actual	Budget
		[columnstyle]		col			col	col
		[row]	row, bold, pad-t-4, row-height-18, pad-B-4		[Format;TargetCell=+1;]	Patient Revenue		
11		[row]	row, brd-tb, brd-color-s2		[Format;ColumnStyle=col, inc	Inpatient	123,456,789	123,456,789

As noted, column styles can also be overridden at the cell level. The following example shows the Format content tag being used to display text using different column styles in each row; in this case to display a set of row labels using different levels of indentation. This approach can be easier than using a bunch of `[ColumnStyle]` rows, especially if you only need to adjust a single cell in the row.



	C	D	E	F	G
2					
3	[Grid;NewTagsGrouped]	[rowstyle]	[fixed]		[Column]
4	[columnwidth]		200px		80px-150px
5	[fixed]	row, bg-s2			[Format;Columns
6	[columnstyle]		col		colhead,right
7	[fixed]	row,bg-s2,row-height-18			2/14
8	[fixed]	row,bg-s2,row-height-18,pad-b-4			Actual
9	[columnstyle]		col		col
10	[row]	row,upper,f3,pad-t-8	[Format;TargetCell=F]	Group1	
11	[row]	row,bold,pad-t-8,f4	[Format;ColumnStyle=col,indent-1;TargetCell=F;]	SubGroup1	
12	[row]	row,bold,pad-t-4,row-height-18,pad-t	[Format;ColumnStyle=col,indent-2;TargetCell=F;]	Patient Revenue	
13	[row]	row,brd-tb,brd-color-s2	[Format;ColumnStyle=col,indent-3;TargetCell=F;]	Inpatient	123,456,789
14	[row]	row,brd-tb,brd-color-s2	[Format;ColumnStyle=col,indent-3;TargetCell=F;]	Outpatient	123,456,789
15	[row]	row,brd-tb,brd-color-s2	[Format;ColumnStyle=col,indent-3;TargetCell=F;]	Other Patient Revenue	123,456,789
16	[row]	row,bold,brd-tb-2	[Format;ColumnStyle=col,indent-3;TargetCell=F;]	Total Patient Revenue	123,456,789

Using base styles and add-on styles

Row and column styles fall into two categories:

- Base styles define the default formatting for the row or column.
- Add-on styles apply specific formatting properties, such as bold font or right-side alignment. These styles are intended to layer on top of a base style, in order to modify that one specific property.

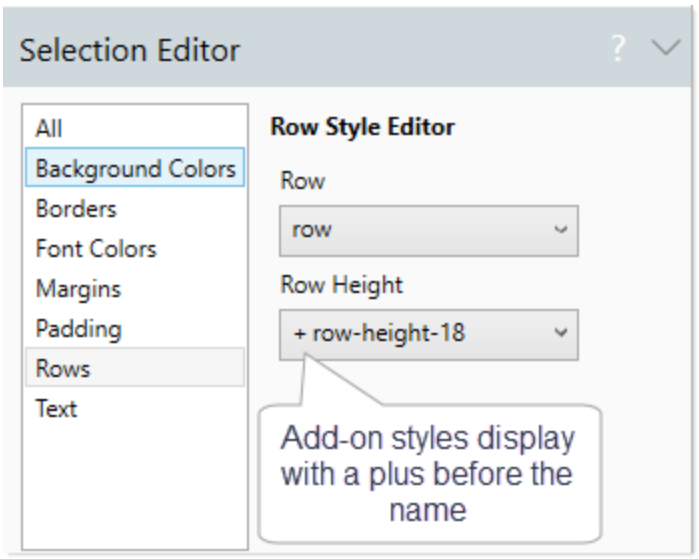
For example, a base style for columns is **col** and an add-on style for columns is **bold**. If you have a column where you want the text to be bold, then you would specify **col, bold** for the style. The second style of **bold** overrides the default font weight for the **col** style, but leaves all the other default formatting intact. If you want to change other aspects of the default formatting, you must use additional add-on styles.

NOTE: If desired, you can omit the base style, and the default base style of either **row** or **col** will be assumed. For example, you can enter **col,bold** or just **bold** into the `[ColumnStyle]` row, and both entries will result in the same formatting because **col** is assumed as the base style. However, if you want to use a different base style, like **colhead**, then it must be explicitly listed.

If your list of styles includes a base style, the base style should *always* be listed first, followed by any number of add-on styles. This is because styles are evaluated in the order they are listed. For example, if you list the styles as **bold,col**, then the column will not be bold because the font formatting in the default **col** style will overwrite the bold formatting applied by the **bold** add-on style. The styles must be listed as **bold** or **col, bold** in order to apply the bold formatting.

In the Data Source Assistant, base styles and add-on styles are differentiated as follows:

- Base styles are listed in the **Row** or **Column** category respectively. However, there are a few additional base styles that can be used for certain types of rows and columns, such as **input-row** for rows with user inputs, and **colhead** for column headers.
- All other styles are add-on styles. These styles display with a plus icon in front of the style name, to indicate that they are applied in addition to a base style.



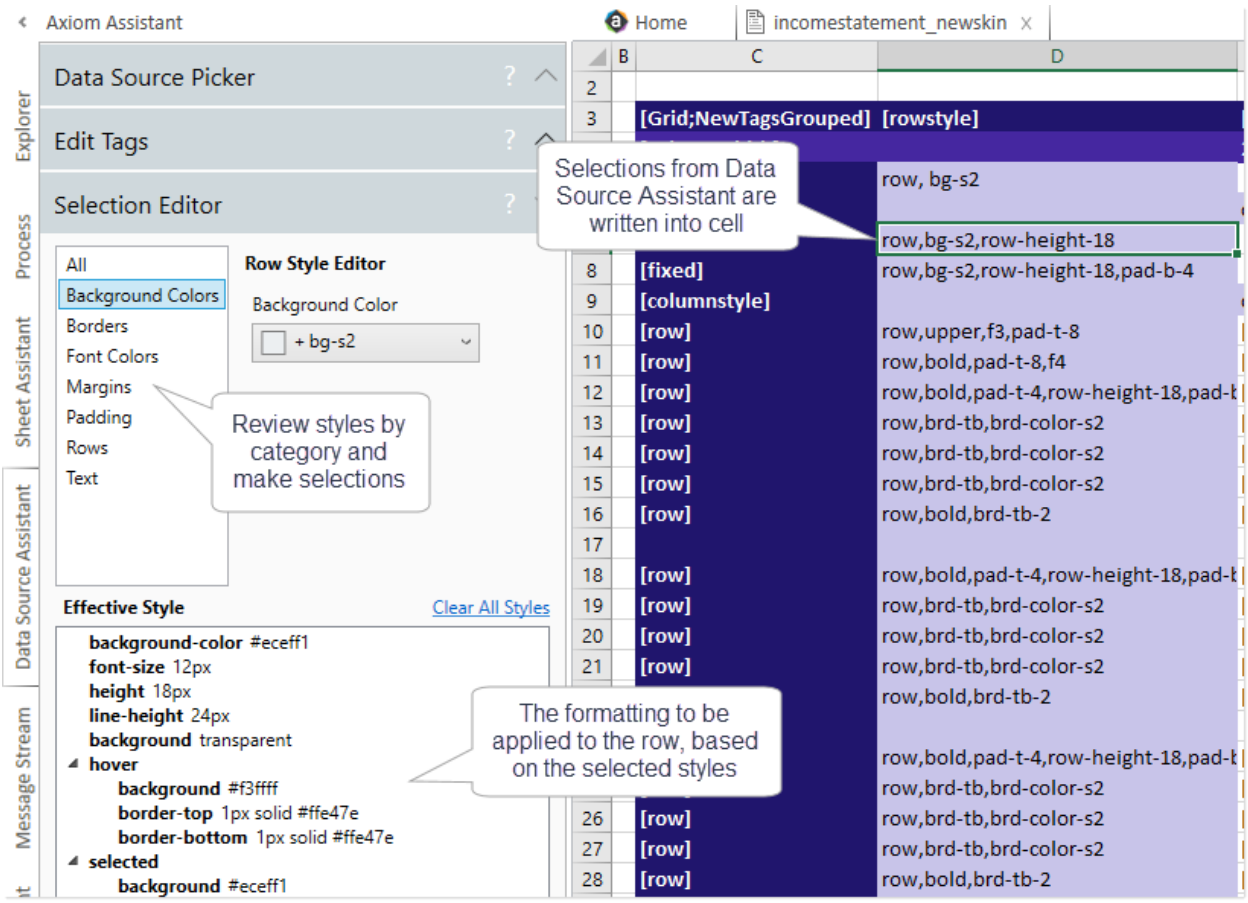
Using the Data Source Assistant to apply row and column styles

You can use the Data Source Assistant to:

- View the available row and column styles
- View the effective formatting applied by each style or combination of styles
- Insert style names into the Grid data source

To view styles and edit style assignments, place your cursor in the `[RowStyles]` column or in a `[ColumnStyles]` row. If you just want to view the styles then the location does not matter; but if you want to edit style assignments then you should place your cursor in the row or column where you want to insert style names or edit existing style assignments.

The Selection Editor section updates to show the available row or column styles. The styles are organized into logical formatting categories, such as Background Colors, Borders, etc. Any style selected from a drop-down list or a check box will be inserted into the cell, using a comma-separated list. If you clear a selection, the style will be removed from the list.



As styles are selected, the effective formatting applied by those styles displays at the bottom of the task pane.

For a full list of available row and column styles, see Thematic grid style reference.

Using theme-based row and column styles (legacy skins)

If your form uses a legacy skin (any skin that is not Axiom2018), the available row and column styles for a grid are different and work differently. There are three main differences:

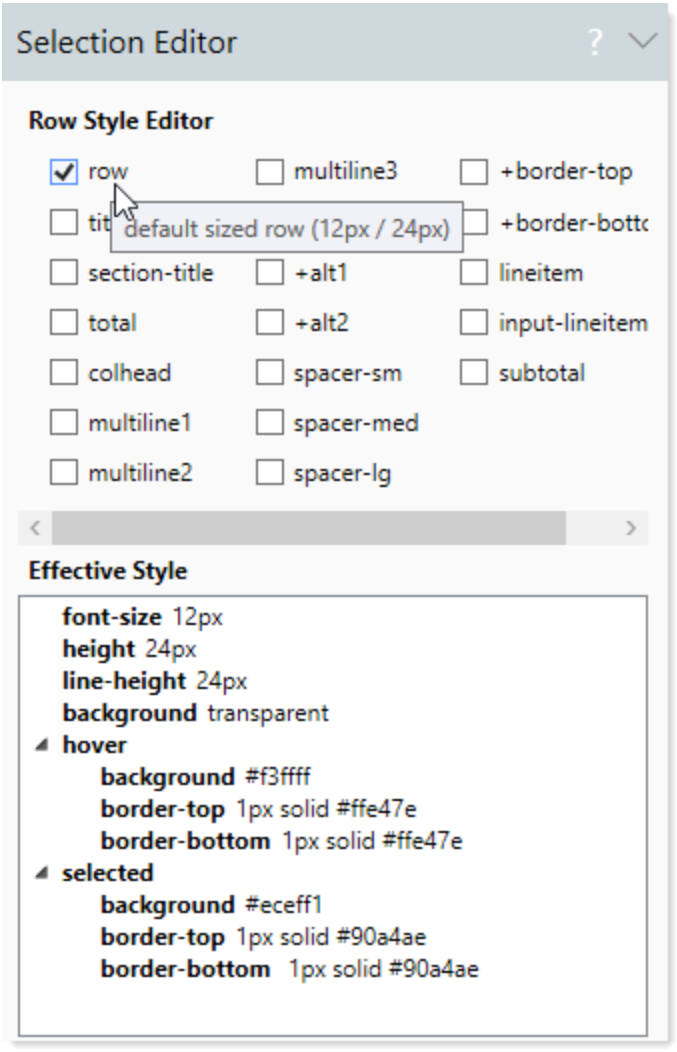
- The available styles depend on the theme specified for the grid. For example, the theme may be Report, Worksheet, or Wizard. This theme can be defined at the form-level and inherited by the grid, or it can be set at the component level. See Setting the theme for a thematic grid (deprecated).

  Most themes support the same core set of styles, though the same style name may apply different formatting in two different themes. However, some themes support certain styles that are only for use in that particular theme. If the theme is later changed and the style name is not found, the row or column will be treated as if no style is applied (meaning the default row or column style for the theme will be used).

- The legacy style names are designed to be semantic. For example, using the legacy approach you would specify a row style named **total** that encompasses all of the formatting for a total row, instead of using the new approach of **row,brd-tb,brd-color-s6,bold** (where the formatting is defined using formatting-specific add-on styles). The legacy approach still uses base styles and add-on styles, and these combine in the same way as described previously.

IMPORTANT: If your form uses a legacy skin, you cannot simply change the skin to Axiom2018. The old row and column style names for grids are not recognized in the Axiom2018 skin. If you want to migrate to the Axiom2018 skin, you must update the row and column styles to use the new format, as described in the previous sections.

When using themes and legacy styles, the style display in the Data Source Assistant and Tag Editor is slightly different. Styles are not organized into categories. Instead, styles display as a series of checkboxes. Add-on styles are still differentiated from base styles by a plus sign in front of the name.



You can select as many styles as needed to achieve the desired formatting for the row or column. Base styles should still be selected first, followed by any number of add-on styles.

NOTE: The Data Source Assistant always displays the styles for the form-level theme. If you have a grid that is assigned a different theme at the component level, the Data Source Assistant will not show those styles. As a workaround, you must temporarily change the form-level theme to the same theme as the component, assign styles to the grid, and then change the form-level theme back to the desired them.

Thematic grid style reference

When using the Axiom2018 skin, thematic grids can use the following row and column styles. These styles provide the ability to set specific formatting properties for each row and column, including fonts, colors, and borders.

NOTE: If the form uses a legacy skin (any skin other than the default Axiom2018), the row and column styles are different. Additionally, the available styles depend on the theme assigned to the form. To see the available styles for legacy skins, use the Data Source Assistant and hover your cursor over a style name for a description of the style. See Using theme-based row and column styles (legacy skins).

Styles are listed by category. Some styles are only for use with rows or columns, while others can be used with either.

When applying styles, remember the following:

- When you include a base style, it must be listed before any add-on styles. When using the Data Source Assistant or Tag Editor to apply styles, select the base style first so that the comma-separated list of styles starts with the base style. Base styles are listed in the **Rows** category for rows, and the **Columns** category for columns. For example: **row, bold, row-height-30**. If you omit the base style, the default **row** or **col** base style is assumed.
- Row styles apply to the entire row, whereas column styles apply until a different column style is used. If a conflict exists between a row style and a column style, the column style takes precedence for the intersecting cell.

Background Colors

You can apply a background color style to rows and columns. Background styles are add-on styles.

Background styles start with **bg-** followed by a color code that indicates a structural, primary, or accent color in the Axiom Software color scheme. For example, **bg-s9** applies structure color 9, which is a dark gray color.

When using the Data Source Assistant or Tag Editor, the background colors are presented in a drop-down list. A color block is displayed next to each code to show the color that will be applied.

Borders

You can apply border styles to rows and columns. You can also specify a color that applies to all defined borders for the row or column. Border styles are add-on styles.

Borders	Style Syntax
Border Top	**brd-t**: Applies a top border of 1px. **brd-t-2**: Applies a top border of 2px.
Border Bottom	**brd-b**: Applies a top border of 1px. **brd-b-2**: Applies a bottom border of 2px.
Border Left	**brd-l**: Applies a left border of 1px. **brd-l-2**: Applies a left border of 2px.
Border Right	**brd-r**: Applies a right border of 1px. **brd-r-2**: Applies a right border of 2px.
Border Presets	**brd-tb**: Applies a top and bottom border of 1px. **brd-tb-2**: Applies a top and bottom border of 2px.
Border Color	**brd-color-s1** through **brd-color-s6**: Overrides the default border color to apply the specified color code. Other Axiom Software color codes are not supported in this context. **brd-color-white**: Overrides the default border color to apply a white border.

Columns

A base column style can be specified for each column. The base style defines the formatting for the column, which can then be further adjusted by adding any number of add-on styles.

The following base column styles are available. See also the Misc category for base styles that fit certain special situations.

Base Style	Description
col	Applies default column formatting. If no base style is specified for a column, the col style is used by default.
colhead	Applies formatting intended for column headers, including a background color. This style is typically used for the first few header cells in the column, and then a second `[ColumnStyle]` row is used to switch the column style back to col.

Font Colors

You can apply font color styles to rows and columns. Font color styles are add-on styles.

Font color styles are simply the Axiom Software color code, such as **s7**.

Indentation

You can apply indentation styles to columns. Indentation styles are add-on styles.

Indentation styles start with **indent-** followed by the indentation level of 0 through 3. For example, **indent-2** indicates that the text should be indented two levels in.

Indentation styles only affect column contents that are left-aligned, either by default or by applying a text alignment style.

Margins

You can apply margin styles to rows. Margin styles are add-on styles.

Borders	Style Syntax
Margin Top	**mgn-t-*Size***: Applies a top margin of the specified size. Supported sizes are 1, 2, 4, 8, or 16 pixels. For example, **mgn-t-4** applies a 4px top margin.
Margin Bottom	**mgn-b-*Size***: Applies a bottom margin of the specified size. Supported sizes are 1, 2, 4, 8, or 16 pixels. For example, **mgn-b-4** applies a 4px bottom margin.

Misc

You can apply miscellaneous styles to columns, to handle specific formatting situations.

Style	Description
auto-width	Sets components in the column to automatic width. For example, if a Button tag is used within the column, the button will be sized based on the button text rather than filling the full width of the column. This style is a base style and can be used on its own.
fit-height	Sets components in the column to fill the height of the row. This is useful for image buttons, to preserve the aspect ratio of the image. This style is a base style and can be used on its own.
click-to-edit	Applies special formatting to unlocked cells. The cells display like locked cells until a user clicks on the cell to edit it. At that point, the cell becomes formatted like a text box. Without this style, unlocked cells display like text boxes. This style is an add-on style.

Padding

You can apply padding styles to rows and columns. For rows, you can apply padding to the top or bottom of the row. For columns, you can apply padding to the left or right of the column. Padding styles are add-on styles.

It is important to understand the difference between margin and padding. Margin is the space outside of an element, whereas padding is the space within an element. For example, imagine that you have a row with a top border. If you apply a top margin of 4px, that margin comes before the top border. The space between the border and the content is unchanged. But if you apply top padding of 4px, the padding comes between the top border and the row content.

Padding	Applies To	Style Syntax
Padding Top	Rows	**pad-t-*Size***: Applies top padding of the specified size. Supported sizes are 1, 2, 4, or 8 pixels. For example, **pad-t-4** applies 4px top padding.
Padding Bottom	Rows	**pad-b-*Size***: Applies bottom padding of the specified size. Supported sizes are 1, 2, 4, or 8 pixels. For example, **pad-b-4** applies 4px bottom padding.
Padding Left	Columns	**pad-l-10**: Applies left padding of 10 pixels.
Padding Right	Columns	**pad-r-10**: Applies right padding of 10 pixels.
Other padding needs	Columns	**pad-noinput**: Applies cell padding that matches the automatic padding applied to input cells. This style is intended to be applied to non-editable cells when you want their padding to match the padding that is automatically applied to editable cells (unlocked cells or cells using TextArea tags).

Rows

A base row style must be specified for each row. The base style defines the formatting for the row, which can then be further adjusted by adding any number of add-on styles. Additionally, you can also specify a row height other than the default row height used by the base style. The following styles are available in this section:

Rows	Description
Row (Base styles)	Select one of the following base styles to apply to the row: • **row**: Applies default row formatting. If no base style is specified for a row, the row style is used by default. • **input-row**: Applies formatting intended for input rows.

Rows	Description
Row Height	Row height styles set a specific row height in pixels, using the syntax **row-height-*Size***. Supported sizes are 18, 30, 33, 55, 70, 100, 200 (all in pixels). Row height styles are add-on styles. **NOTE:** If you need to set the row to a custom height, you can use the `[RowHeight]` column in the data source instead. If the height is set in the `[RowHeight]` column, it overrides the value set by the style. See Setting row sizes for Formatted Grids.

Text

You can apply various text styles to rows and columns. Most text styles are add-on styles.

Text Format	Style Syntax
Text Alignment	The following add-on styles can be used to control text alignment: • **left** • **center** • **right** If text alignment is not specified, the default alignment is applied.
Font Size	Font size can be set using the add-on styles **f1** through **f7**, where f1 is the largest font and f7 is the smallest. See the Effective Style details for the exact font size.
Weight	The following add-on styles can be used to control font weight: • **light** • **bold**
Transform	The following add-on styles can be used to transform the text case (upper-case or lower-case): • **lower** • **upper**
Other	The following additional add-on styles can be applied to format text: • **underline** • **italic** • **wrap-text** (column style only) The following additional base styles can be used for specific formatting situations: • **underlined** (column style only): This style causes hyperlink text to be formatted with blue font and an underline. Non-hyperlink text is unaffected.

Setting the theme for a thematic grid (deprecated)

IMPORTANT: Themes only apply to legacy skins. If your form uses the default Axiom2018 skin introduced in version 2018.1, themes do not apply.

Thematic grids depend on a specified theme to determine general formatting properties for the grid, as well as to determine which row and column styles are available to be used in the data source.

By default, the Formatted Grid component inherits the form-level theme, just like all other components in the form. However, while other components may render just fine if the form does not have a specified theme, thematic grids almost always require a theme. Therefore, you should always make sure to specify a form-level theme if the form contains a thematic grid. To do so, click **Edit Form Properties** in the Form Designer or Form Assistant, and then use the **Theme** property to assign a theme. For more information on specifying the theme for a form, see Setting the theme for an Axiom form (deprecated).

Because thematic grids are very dependent on the theme, it is possible that you may want to use different themes for multiple grids in the same form. For example, you may have a form where you want one grid to use the Report theme and the other grid to use the Worksheet theme. Whichever theme is best suited for the overall form contents should be set at the form level, and the grids will inherit that theme by default. For the grid that you want to use a different theme, you can override the form theme at the component level by using the **Component Theme** property in the advanced component settings (click **Show Advanced Settings** under the **Style** box). The Component Theme property is available for all components, but it is most likely to be used on thematic Formatted Grids.

For grids, some of the differences between themes include:

- The Wizard theme and Worksheet theme have more row padding, to account for spacing between multiple rows of input controls.
- The Report and Grouped Report themes have less row padding, to allow displaying many rows of data on a single page.
- The Report and Grouped Report themes provide row and column styles for typical report designs, such as column headers, and total and subtotal rows.
- The Grouped Report theme provides additional row and column styles to display multiple levels of headings (groupings) in a single report.

Using spreadsheet formatting for a Formatted Grid

When the **Grid Formatting** option for a Formatted Grid component is set to **Spreadsheet**, then the grid adopts the formatting defined in the spreadsheet. You must format the cells in the data source as you want them to display in the rendered form. This includes setting formatting such as fonts, colors, and borders.

IMPORTANT: The spreadsheet formatting option primarily exists to support backward-compatibility for forms that were created prior to the introduction of the thematic grid (in version 2016.1). For any new grids, we strongly recommend using the thematic grid, as it will be the focus for all new enhancements moving forward.

The adoption of the spreadsheet formatting in the Axiom form is as close as possible, but some limitations and exceptions apply. The following table summarizes the spreadsheet formatting options and how they apply to the formatted grid in the Axiom form:

Spreadsheet Format	Notes and Limitations
Borders	Inherited as defined in the spreadsheet.
Column width	Column width can be set using the spreadsheet column widths, or by using the `[ColumnWidth]` tag. When using the spreadsheet column widths, the actual width in the form depends on whether **Fit Columns** is enabled for the Formatted Grid component. For more information on both options, see Setting column sizes for Formatted Grids.
Conditional formatting	Conditional formatting can be used with some limitations. See Using Conditional Formatting in spreadsheet-formatted grids.
Data validation	The following types of Data Validation are supported: • **Drop-Down lists**: You can use the **List** option of Data Validation to define drop-down lists for use in the formatted grid. • **Number validation**: You can use the **Decimal** option of Data Validation to determine valid numeric inputs for the cell in the formatted grid. For more information, see Using Data Validation in spreadsheet-formatted grids.
Fill formatting	• Fill colors are inherited as defined in the spreadsheet, unless content tags are used to specify the background color. For more information, see Using content tags in Formatted Grids. • Fill patterns are *not* supported.
Font formatting	Inherited as defined in the spreadsheet, with one exception: if content tags are used to specify the foreground color, this overrides the font color. For more information, see Using content tags in Formatted Grids. It is strongly recommended to use a common font such as Arial, which all client machines and devices are likely to support. If the font used in the spreadsheet is not found on the client machine, then the font specified by the form-level skin is used. **NOTE:** Underlined text is not supported by Mozilla Firefox.
Number formatting	Inherited as defined in the spreadsheet.

Spreadsheet Format	Notes and Limitations
Protection	The cell Locked status is used to determine whether a cell in the grid is editable or not. The sheet does not have to be protected in order for this setting to take effect in the Axiom form. For more information, see Interactivity options for Formatted Grids.
Row height	Inherited as defined in the spreadsheet. Formatted Grid components support options to auto-expand or auto-shrink the grid to match the total height of all rows in the data source. For more information, see Setting row sizes for Formatted Grids.
Text alignment	Inherited as defined in the spreadsheet.
Text controls	• Wrapped text is supported, but should only be used when absolutely needed due to performance considerations. If all cells in a grid are set to wrap text, this may impact form performance. • Merged cells are *not* supported. If the cell is for display only, you can use Center Across Selection instead. If the cell is for user input, you can use the TextArea content tag instead (see Using text boxes in Formatted Grids).

Additional considerations

Note the following additional considerations about formatted grids:

- If you want to display a symbol in the grid, use the special content tags to specify the symbol rather than simply changing the cell font to something like Wingdings. Wingdings and other built-in symbol fonts do not display on non-Windows platforms (such as when viewing Axiom forms via the iPad). For more information, see Using symbols in Formatted Grids.
- You can include hyperlinks in a grid in a variety of ways. Note however, that you cannot simply enter a plain hyperlink into the cell. For more information on the various options, see Using hyperlinks in Formatted Grids.
- If an Initial Dynamic View is set for the source file, it will be applied when the file is viewed as a form and can be used to impact the row heights and column widths of Grid data sources. The view will also be reapplied after Axiom queries are refreshed in the source file. However, there is no way to change the view from an Axiom form—once it is applied as the Initial Dynamic View it cannot be changed or removed.
- Raw HTML tags placed in the grid will not be rendered in the Axiom form.
- When the file is rendered as an Axiom form, the source file is always read using the Web spreadsheet engine. Therefore, any spreadsheet design considerations that apply to the Windows Client also apply to use of Axiom forms. For more information, see the *Axiom File Setup Guide*.

Using Data Validation in spreadsheet-formatted grids

Certain features of Microsoft Excel's Data Validation are supported for use in Formatted Grid components:

- **List**: You can use the List option of Data Validation to create a drop-down list in a formatted grid.
- **Decimal**: You can use the Decimal option of Data Validation to define valid numeric inputs in a formatted grid.

To set up these options, configure the cell using Excel's **Data Validation** feature as normal (from the **Data** tab of the Excel ribbon). The specific configuration requirements for use in Axiom forms are detailed below.

If you are using the Axiom Windows Client instead of the Excel Client to configure the form, you can access Data Validation features by right-clicking the cell and selecting **Range Explorer**. Within the Range Explorer dialog, click **Validation**. The settings are similar to those presented in Excel.

Using Data Validation to create a drop-down list

You can use the **List** option of Excel's Data Validation feature to create a drop-down list of choices within a Formatted Grid component.

- For the list **Source**, you can specify a comma-separated list of values, or you can use the Indirect function to point to a range (for example: `Indirect("Info!A1:A10")`. The Indirect function must be used with all ranges, whether on the same sheet or cross-sheet.
- The cell that is configured for Data Validation must be unlocked so that it will be editable for the form user.

When the form user clicks on the cell configured with Data Validation, a drop-down list will present the list of valid values. The user can select an item from the list. The selected item is submitted back to the source file and placed in that cell.

Keep in mind that until the user clicks on the cell, no visual cues are present to tell the user that the cell is editable. The list arrow does not display until the user clicks on the cell. You may want to format the cell using a convention that indicates the cell is editable (such as a yellow background), and/or place text in the adjacent cell (such as "Select a category >>").

NOTE: The Excel Data Validation feature should only be used for simple lists that do not relate to data stored in Axiom Software. The Select content tag should be used for most drop-down lists in formatted grids, due to the many additional features supported by the tag, and for the ability to source the list from Axiom Software data. For more information, see Using drop-down lists in Formatted Grids.

Using Data Validation to validate numeric inputs

You can use the **Decimal** option of Excel's Data Validation feature to define valid numeric inputs for cells of a formatted grid. For example, you can specify that the input must be greater than or less than a

specific value, or that the input must be within a range of values. This validation is performed before any updates are submitted to the source file.

The following rules apply when setting up numeric Data Validation:

- Only the Decimal option is supported in Axiom forms for purposes of validating a non-list input. The options for whole number, date, etc. are not supported and will be ignored.
- The validation values can be constant or can use a formula cell reference to read the values from the source file. If you use a cell reference, it is recommended to reference a cell within the current sheet. If you must use a cross-sheet reference, the reference must be placed within an INDIRECT function. Note that the **Ignore blank** setting is ignored in all cases.
- A custom error message must be defined on the **Error Alert** tab of Data Validation. If no custom error message is defined, then the validation will not be enforced in the Axiom form. Note that the alert Style setting is ignored; all validation errors display in the same message style within Axiom forms.
- The **Input Message** settings are not supported in Axiom forms and will be ignored.

When the form user exits an editable cell where Data Validation is defined (for example, by clicking out of the cell, or by pressing the Enter or Tab keys), the validation rules are applied to the cell contents. If the contents are invalid, the custom error message displays in a message box. The user can click **OK** in the message box or press the Esc key to dismiss the message and be returned to the cell. The user can then change the value to a valid value, or they can press the Esc key to revert to the previous cell value.

NOTES:

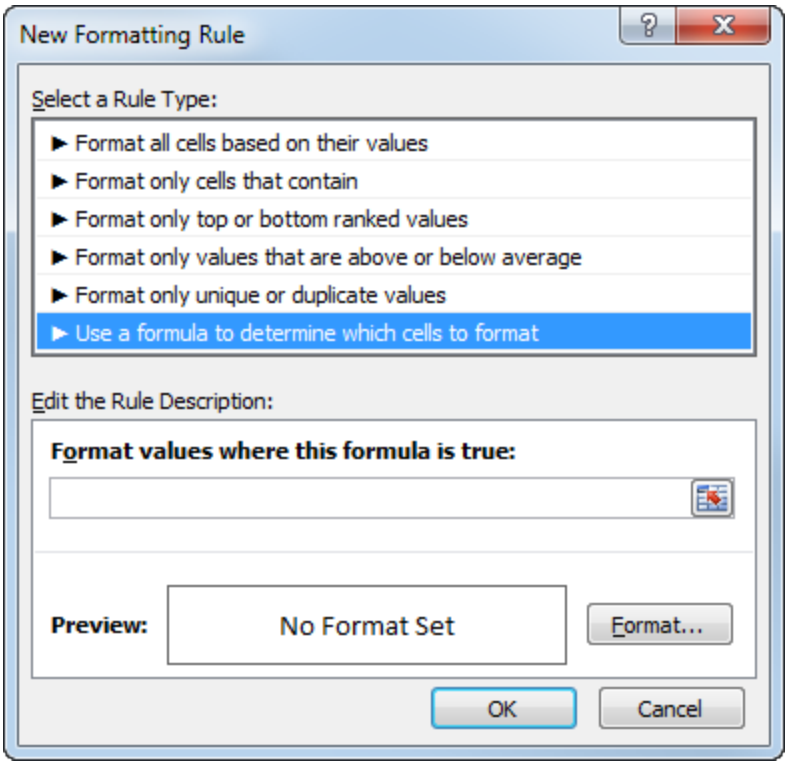
- Validation is only performed when the cell is edited. If the cell already contains an invalid value, it is not validated until you attempt to edit that value.
- When using tablets, there is no way to revert to the previous cell value once it has been edited.
- Custom save validation can be used as an alternative to Data Validation. Using custom save validation, you define conditions to be checked as part of the save-to-database process. If a value does not meet the defined condition, then the save is stopped and an error displays to the user. For more information, see the Axiom File Setup Guide.

Using Conditional Formatting in spreadsheet-formatted grids

You can use the spreadsheet Conditional Formatting feature in Formatted Grid components, with some limitations. This is only available when the grid is configured to use spreadsheet formatting. Thematic grids will ignore the spreadsheet formatting, including any conditional formatting.

When setting up the conditional formatting in the spreadsheet, keep in mind the following:

- The condition must use a formula. This means that when you define the formatting rule in Excel, the rule type must be **Use a formula to determine which cells to format**. If any of the other rule types are used, an error will result when the form is rendered.



- The colors specified for the conditional formatting may not display in the Axiom form exactly as defined within the spreadsheet. You should test the form and make adjustments as needed to achieve the color that you want.

  NOTE: If you need to specify the color exactly, then instead of using Excel's conditional formatting you can define your own conditional formula that dynamically displays a content tag to impact the cell colors. For more information, see Using content tags in Formatted Grids.

- The same conditional formatting limitations that apply to the Windows Client also apply to use in Axiom forms. For example, the following conditional format configurations are not supported: cross-sheet references, overlapping formats, and more than three formats per cell.

Migrating spreadsheet-formatted grids to thematic grids

If you have Formatted Grid components in existing forms, it is recommended to migrate them from using the previous spreadsheet formatting to the new thematic formatting. Thematic formatting has many advantages over spreadsheet formatting, such as:

- The grid formatting is now externalized from the spreadsheet, making it easier to apply consistent formatting across many files (by using the same styles). Additionally, if the formatting defined in the style changes, the change automatically flows through to all grids using the style.
- The grid formatting can now exactly match the formatting used in the rest of the form, since the skin formatting now applies to the grid contents. Additionally, grid features such as combo boxes and check boxes use the same controls as the stand-alone components.
- It is easier to implement additional features in a format that does not need to also support spreadsheet formatting. Because this restriction is removed for thematic grids, these grids support new features such as the ability to display a date picker or a button within the grid. The thematic format will be the focus for all future grid enhancements going forward.

The ease of the migration process depends on the grid contents, and the size of the grid. Simple grids with only a few unique formatting elements will likely be quick and easy to migrate. Larger grids with many formatting elements and/or heavy use of content tags may be more time consuming to migrate.

Migration considerations

Although thematic grids have many advantages over spreadsheet-formatted grids, there are a few features that they do not support. Keep in mind the following:

- The numeric Data Validation feature is not supported in thematic grids. Although it is possible to design alternative ways to communicate out-of-bounds values to users, and to prevent saving data, it is not possible to prevent inputs at the point of entry.
- Thematic grids do not support the Conditional Formatting feature, nor do they support the Foreground and Background parameters in content tags. The ColumnStyle parameter for thematic grids is potentially more powerful and effective to conditionally control formatting via formula, but currently you are limited to using the built-in styles (since customizing styles is not officially supported in this release). The built-in styles may not cover the types of conditional formatting changes that you want to make in the grid.
- Unlocked cells in thematic grids display as editable text boxes, meaning that the cells are bordered like a text box. If you want unlocked cells to display in a similar manner as the spreadsheet grid—where the cell contents display as normal cells until the user clicks in the cell to edit—then you can use the `CLICK-TO-EDIT` column style to apply this behavior. The Web Client container must be enabled to use this style.

- Thematic grids do not accommodate text that exceeds the designated column width. If a cell has overflow text, the overflow text will be cut off. This is in contrast to spreadsheet-formatted grids, where overflow text displays in the adjacent cell if the adjacent cell is blank. You can handle overflow text in a thematic grid using one of the following approaches:
    - You can use a Format tag to span the cell contents across multiple columns. To do this, you would need to move the cell contents to a target cell, and then insert a Format tag into the cell where you want the contents to display. You can then configure the tag to span as many columns as needed to display the contents. For more information, see Applying formatting to cell contents in Formatted Grids.
    - You can wrap the text so that it displays as multi-line content within the cell. To do this, you must set a column style that includes wrap-text, and also make sure that the row style is tall enough to display multiple lines.

Preparing the form

If the form is actively being accessed by end users, it is recommended to create a copy of the form in which to perform the migration. Once you have made all changes and tested the form, you can replace the old form with the new form.

Within the migration copy, you should do the following to prepare the form for the migration:

- In the Form Properties, set the **Skin** property to **Axiom2018**. For more information, see Setting the skin for an Axiom form.

It is not required to switch the skin to Axiom2018 in order to use a thematic grid, but it is strongly recommended to do this as part of updating the form to use the latest design elements and features. However, if necessary, you can continue to use the existing legacy skin and just update the grid from spreadsheet to thematic. Keep in mind that if the form uses a legacy skin, this means you must also use the legacy grid styles instead of the new styles. Also, if the form does not currently have an assigned theme, you should assign the theme that most closely fits the purpose of your form.

Migrating the grid

To migrate the existing spreadsheet grid to a thematic grid, do the following:

1. In the Formatted Grid properties, change **Grid Formatting** from **Spreadsheet** to **Thematic**.
2. Go to the data source for the grid (you can click  next to the data source name in the grid properties), and then add the missing thematic tags to your data source. You must add the following:

Tag	Description
[RowStyle]	It is recommended to place this tag in the cell directly to the right of the primary [Grid] tag, before any content columns. This placement is not required, but will make it easier to read the settings used by the grid going forward. If desired, you can place this tag anywhere in the control row (to the right of the primary tag).
[ColumnWidth]	It is recommended to place this tag in the cell directly below the primary [Grid] tag, before any content columns. This placement is not required, but will make it easier to read the settings used by the grid going forward. If desired, you can place this tag anywhere in the control column (below the primary tag).
[ColumnStyle]	It is recommended to place this tag in the cell directly below the [ColumnWidth] tag (or directly below the primary [Grid] tag, if the width tag is elsewhere). This tag must be placed before any content rows, because it defines the initial style to be used for the columns. If any content rows are above the first [ColumnStyle] tag, then the columns in those rows will use the default column style or the column style specified per cell within content tags. You may need to insert additional [ColumnStyle] tags to change formatting within the content, but for now you can just insert the tag to define the starting column style.

You can add these tags manually, or you can use the Data Source Assistant to add the tags. If you place your cursor within the data source and click **Insert** for a tag, Axiom Software will insert a new column to the left of the current location or a new row above the current location.

The following example shows a data source with the recommended placement of tags.

	A	B	C	D	E	F	G	H
9								
10			[Grid;Thematic]	[RowStyle]	[Fixed]	[Fixed]	[Column]	[Column]
11			[ColumnWidth]					
12			[ColumnStyle]					
13			[Fixed]		Content	Content	Content	Content
14			[Fixed]		Content	Content	Content	Content
15			[Row]		Content	Content	Content	Content
16			[Row]		Content	Content	Content	Content

Don't worry about populating the new rows and columns at this point; this will be discussed in later steps.

NOTE: If you insert columns and rows, be aware of any breaking changes this may have on your form. You may have other components that reference specific cells in this sheet, and you may have content tags inside this grid that reference specific cells. For example, if you have a Select tag with a target cell of N, this may now need to point to column O after inserting a column.

3. In the `[ColumnWidth]` row, set the column width for each column. You can set the width in pixels or percentages. For more information, see Setting column sizes for Formatted Grids.

   Note the following:

   - If you previously had blank "spacer" columns in your grid, you may not need them any more. Once you have all of the row and column styles set up, try clearing out the column tags for these spacer columns and preview the form. Depending on what content is being displayed in the grid and which styles you are using, the columns may have enough padding. If not, you can add the spacer columns back as needed and set them to a very specific size, like 10px.
   - Fit Columns does not apply to thematic grids. If you want to make sure that the grid contents fill the component width, you can use percentage sizing, or leave one or more columns blank so that they are auto-sized to fill the remaining width. (However, it is not recommended to leave all columns blank, as that will result in all columns being the same size.)

4. In the `[RowStyle]` column, set the style for each row.

   The easiest way to do this is to use the Data Source Assistant. Place your cursor in the column to see a list of all available styles and assign them to each row. For more information, see Using row and column styles in a thematic grid.

   Note the following:

   - Among other things, the row style sets the row height. If you have a row that needs to be taller than other rows—for example, because it allows for multi-line input or wrapped text—then you must apply a style that sets a taller row height. Row height styles are set in the **Rows** category.
   - If you previously had blank "spacer" rows in your grid, you may not need them any more. Once you have all of the row and column styles set up, try clearing out the row tags for these spacer rows and preview the form. Depending on what content is being displayed in the grid and which styles you are using, the rows may have enough padding. If not, you can add the rows back as needed, and apply a spacer style to those rows.

5. In the `[ColumnStyle]` row, set the initial style for each column.

   The easiest way to do this is to use the Data Source Assistant. Place your cursor in the row to see a list of all available styles and assign them to each column. For more information, see Using row and column styles in a thematic grid.

Depending on the grid contents, you may find that you need to apply a different style to certain rows in the column, or to specific cells in the column. You can apply column styles to specific cells when using format tags, and you can add as many `[ColumnStyle]` rows as needed to change column styles.

6. If you are using content tags in the grid (for example, Select tag, Checkbox tag, Symbol tag), in many cases these tags will work as is and do not require any further migration. However, note the following special considerations:
    - If you are using the Foreground or Background parameters in any tag, these parameters are not supported for use in thematic grids. Instead, you must use the ColumnStyle parameter to set the foreground color or background color via styles.
    - If you are using the CheckBox tag with the NoAutoSubmit parameter, this must be converted to the AutoSubmit parameter for thematic grids.
    - If you are using the AddRows tag, this tag is not supported for use in thematic grids. Instead, you must use a Button tag and set the command to the AddRows command. The setup is otherwise the same. If you want the button to look like link text instead of a push button, set the ButtonStyle parameter to Link.
7. Make sure that all cells in the data source are locked, unless you deliberately want them to be unlocked for text input. Keep in mind that unlocked cells will display as editable text boxes in thematic grids. You may want to switch these cells to using the TextArea tag to provide better control over the display of the text box (such as the ability to apply column styles to the cell).
8. If the spreadsheet grid used either Data Validation or Conditional Formatting, these spreadsheet features are not supported in thematic grids. You will need to convert these features as follows:
    - Data Validation for the purposes of displaying lists should be converted to using the Select tag with a ComboBox data source.
    - Data Validation for the purposes of validating numeric entries is not currently supported in thematic forms. As an alternative, you can use an approach where you test values using a formula and display an error message to the user in a designated location if a value is out of bounds. You can configure the form so that saving data (or performing some other action) is not available if any of these error messages are present.
    - Conditional Formatting should be converted to changing column styles via formula. For example, column styles can be used to dynamically display a cell in bold, or in green font, or in red font.

You will likely need several rounds of viewing the form and tweaking settings to get the grid to display as you want it. You may need to experiment with column sizes and the effects of various row and column styles. Make sure to test all content tags to ensure these are working as expected (especially if columns have been added or removed from the sheet).

Using content tags in Formatted Grids

Formatted grids support a variety of content tags that can be used to provide interactive controls and display features within the grid. Using content tags, you can include the following features within a formatted grid:

- Searchable drop-down lists (Select tag)
- Interactive check boxes or toggle switches (CheckBox tag)
- Text boxes (TextArea tag)
- Date pickers (DatePicker tag)
- Command buttons (Button tag)
- Formatted display text (Format tag)
- Symbols (Symbol tag)
- Sparkline charts (Sparkline tag)
- Hyperlinks to various documents, including other forms (HREF tag)

To use these features, you place the designated tag within the cell where you want the feature to display. Each tag has a variety of parameters to configure the behavior and appearance of the feature. Within the spreadsheet, you will only see the tag. But when the form is rendered, the tag and its parameters will be converted to the desired feature for display to form users.

Content tags can be used in both spreadsheet-formatted grids and thematic grids. However, certain tags and/or parameters may only be available in one type of grid or the other. Additionally, the display of certain features may be slightly different in each type of grid.

Using buttons in Formatted Grids

You can use the Button content tag within a Grid data source to present an interactive button to users.

The Button tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the Button tag will be resolved as an interactive button. The user can click the button to trigger a form update and optionally perform a designated command.

NOTE: The Button tag is only supported for use in thematic grids. If a Button tag is present in a spreadsheet-formatted grid, it will display as the raw tag text.

Content tag syntax for buttons

The syntax for the Button content tag is as follows:

```
[Button; Text=LabelText; Tooltip=Text; IsEnabled=True/False;
ButtonBehavior=BehaviorName; TargetDialogPanel=ComponentName;
TargetFormattedGrid=ComponentName; ButtonStyle=StyleName; ImagePathURL=Path;
Symbol=SymbolName; SymbolPosition=Left/Right; SaveOnSubmit=True/False;
```

```
ConfirmationMessage=MessageText; Command=CommandString;
TargetDialogPanel=ComponentName; Columns=Number; ColumnStyle=StyleName]
```

Parameters can be listed in any order after the Button tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

Parameter	Description
Text	The text to display on the button. You can define the button text within the tag directly, or you can use a bracketed cell reference to read the text from another cell. For more information, see Referencing cells in content tag parameters. **NOTE:** The Text parameter does not apply if the ButtonStyle is set to Image. You can also optionally omit the button text if a symbol is specified for the button.
ButtonBehavior	Optional. The button behavior to use for the button. • **Command** (default): Updates the form and can optionally execute a command. • **ShowDialogPanel**: Opens a dialog defined by a designated Dialog Panel component. A command can still be optionally executed when using this behavior. • **EditGridDatainSpreadsheet**: Opens the contents of a specified Formatted Grid component in a spreadsheet-style editor. These button behaviors are the only behaviors currently supported by the Button tag. The other button behaviors that are available for the Button component cannot be used by the Button tag. For more information on button behaviors, see Button behaviors.
TargetDialogPanel	The name of the Dialog Panel component to open when the button is clicked. This parameter only applies when using the ShowDialogPanel button behavior. For more information on using Dialog Panel components, see Dialog Panel component.
TargetFormattedGrid	The name of the Formatted Grid component to open in the spreadsheet editor when the button is clicked. This parameter only applies when using the EditGridDatainSpreadsheet button behavior. For more information on using the spreadsheet editor, see Editing grid contents in a spreadsheet editor.

Parameter	Description
SaveOnSubmit	Optional. Specifies whether a save-to-database occurs when a form update is triggered by this button. This parameter only applies when the button behavior is Command. By default this parameter is False, which means no save-to-database will occur. If this parameter is set to True, then a save-to-database will occur as part of the update process. This save occurs after editable values have been submitted to the source file and after data has been refreshed in the source file. The file must be configured to enable a save-to-database process. For more information, see Saving data from an Axiom form.
IsEnabled	Optional. Specifies whether the button is enabled. If omitted or True, the button is active. If False, the button is grayed out and inactive. Generally speaking, the only reason not to use True is if you are dynamically enabling and disabling the button based on some other factor. For example, you may be checking to make sure that all required inputs have been made before enabling the button. In this case, you must construct the tag using a formula so that you can change the value of this parameter.
ButtonStyle	The style of button. Specify one of the following: • **Push** (default): The button displays as a standard rectangular button. The user clicks the button to perform the button action. • **Link**: The button displays as if it is a hyperlink. The user clicks the link to perform the button action. The button text defines the hyperlink text. • **Image**: The button displays as an image. The user clicks on the image to perform the button action. When using this option, you must specify either an image file or a symbol to use as the button image. Button tags support the same button styles as the Button component. For more information and examples of the different button styles, see Using different button styles.

Parameter	Description
ImagePathURL	The image to display for the button. This parameter only applies when ButtonStyle is set to Image. Specify the full path to the image in the Axiom file system, for example: `\Axiom\Reports Library\Images\axiom_logo.png` You can define the path within the tag directly, or you can use a bracketed cell reference to read the path from another cell. For more information, see Referencing cells in content tag parameters. **NOTES:** • If the image is later moved or renamed, you must edit the tag to specify the new name or location, otherwise the image reference will be broken. This path does not automatically update. • Users must have permission to the image file in order to see it rendered in the form. It is recommended to create a dedicated Images folder in the Reports Library and store all images in this location. You can grant access to this folder using the Everyone role, or you can create subfolders and grant access to users and roles as needed. • You can specify either an image path or a symbol for the image button. If you specify a symbol, then the Image Path fields are hidden in the Tag Editor / Data Source Assistant. If you originally specified a symbol but now you want to specify an image path, you must first clear the symbol in order to make the Image Path fields available again.

Parameter	Description
Symbol	Optional. The symbol to use for the button. The symbol applies as follows: • For push and link buttons, the selected symbol displays on the button in addition to the button text. You can also optionally omit the text when a symbol is specified. • For image buttons, you can optionally use a symbol for the button image instead of specifying an image file. In the Tag Editor / Data Source Assistant, click the **[...]** button to open the **Choose Symbol** dialog. Within this dialog, you can scroll through the available symbols, or you can use the filter box at the top to find symbols based on symbol names. For example, you can type `file` to see all of the symbols that have the word "file" in the name. When you have found the symbol that you want to use, select it and then click **OK**. The selected symbol shows in the Tag Editor / Data Source Assistant, and the actual symbol name is written to the Button tag. Alternatively, you can use a bracketed cell reference to read the symbol name from another cell. For more information, see Referencing cells in content tag parameters. **NOTE:** If you select an image path for an image button, then the Symbol fields are hidden. If you originally selected an image path but now you want to select a symbol, you must first clear the selected image path in order to make the Symbol fields available again. (If you specify both an image path and a symbol by manually editing the Button tag, the symbol takes precedence.)
SymbolPosition	The position of the symbol relative to the button text (Left/Right). Left is the default position. This parameter only applies to push and link buttons, and only if a symbol is specified for the button.
ConfirmationMessage	Optional. Defines a confirmation message to display before performing any button actions. This parameter only applies when the button behavior is Command. If a confirmation message is defined, then when a user clicks the button in the Axiom form, a message box will display the message. The user can click **OK** to proceed with the button actions, or click **Cancel** to abort the form update and any assigned command. You can define the message within the tag directly, or you can use a bracketed cell reference to read the message from another cell. For more information, see Referencing cells in content tag parameters.

Parameter	Description
Command	Optional. Specifies one or more commands to perform when the button is clicked. The same commands that are available for the Button component can also be used here. For more information, see Using buttons to perform commands. The command to execute and its parameters must be placed in a valid command string. This string looks something like the following: `command://ProcessActionCodesCommandAdapter?sheet=Sheet 2&tag=Action&_ps=AfterUpdateValues` However, it is not recommended to manually create your own command string. Instead, you should use the Tag Editor dialog or the Data Source Assistant to select the desired command and define the parameters, and then it will create the string for you. You can optionally define multiple commands. When using the editor, click Add to add another command. Multiple commands are added as comma-separated strings to the Command parameter. You can define the command within the tag directly, or you can use a bracketed cell reference to read the command from another cell. For more information, see Referencing cells in content tag parameters. Note that if you use a cell reference, each cell must contain a single command. If you want to use multiple commands, you must use multiple cell references separated by commas (or a command string plus a cell reference). For example: `Command=[MySheet!A1],[MySheet!A2]`

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Button behavior notes

- Push buttons will either extend to fill the column width or auto-size based on the button text, depending on the button width property of the assigned style. For example, the **col** style will cause the button to fill the column width, while the **auto-width** style causes the button to auto-size.

- Link buttons are treated as hyperlink text for sizing and positioning purposes. The text-align property defined for buttons is ignored; instead the general text-align property applies. For example, apply a style that includes centered text positioning if you want the text to be centered in the column.

  By default, the hyperlink text is in blue font and without an underline. If desired, you can use the following column styles to change the display:

  - **text-color**: Removes the blue color and instead displays in the default text color.
  - **underlined**: Applies an underline to the text.
- If you are using an image button with an image file, you should apply the **fit-height** column style (or another style with button height at 100%). This will cause the image to scale to fit the row height, while maintaining the size ratio of the image.
- Grid buttons and the Apply Form State command cannot be used in a refresh form (meaning when an Axiom form serves as the refresh dialog in the Desktop Client). In this use case, the Apply Form State command must be combined with the Close Dialog command in order to trigger the refresh behavior. Because Button tags can only use a single command, a stand-alone Button component must be used instead. For more information, see Using an Axiom form as a refresh form.

Examples

```
[Button; Text=Refresh; ButtonBehavior=Command; ButtonStyle=Push]
```

This example is a basic button that triggers a form update.

```
[Button; Text=Action codes; ButtonBehavior=Command; ButtonStyle=Link;
Command=command://ProcessActionCodesCommandAdapter?sheet=Sheet2&tag=Action&_
ps=AfterUpdateValues]
```

This example executes the Process Action Codes command in addition to the normal form update. The button style is Link so the button will display as link text instead of as a push button.

```
[Button; ColumnStyle=fit-height; ButtonBehavior=Command; ButtonStyle=Image;
ImagePathUrl=\Axiom\SystemFolderName_ReportsLibrary\Images\new_axiom_logo.png]
```

This example displays as an image button using the specified image. A style is applied at the tag level so that the image will be auto-sized to fit the current cell.

```
[Button; Text=Save; ButtonBehavior=Command; SaveOnSubmit=True;
ButtonStyle=Push; ConfirmationMessage=Are you sure you want to save this
data?]
```

This example triggers a save-to-database in addition to the normal form update. The defined confirmation message will display before any action is taken, to give the user a chance to cancel if desired.

```
[Button; Text=Save; ButtonBehavior=Command; SaveOnSubmit=True;
ButtonStyle=Push; Symbol=fa-save; SymbolPosition=Right;]
```

This example displays a symbol on the button in addition to the text. The symbol displays to the right of the text.

```
[Button; Text=Add Row; ButtonBehavior=ShowDialogPanel;
TargetDialogPanel=AddRowDialog; ButtonStyle=Push;]
```

This example uses the ShowDialogPanel behavior to open a specified Dialog Panel component as a dialog.

```
[Button; ButtonBehavior=Command; Text=OK; ButtonStyle=Push;
Command=command://ApplyFormStateCommandAdapter?refreshMode=Worksheet,
command://CloseDialogCommandAdapter;]
```

This example can be used in a form dialog to apply form state to the active client spreadsheet and then close the dialog.

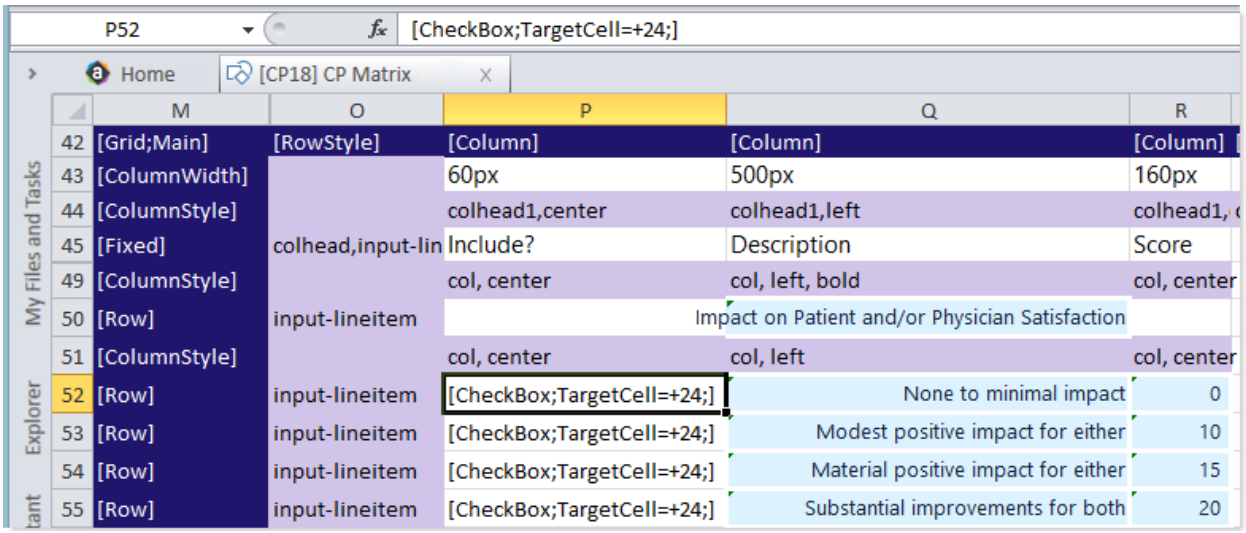
Using check boxes in Formatted Grids

You can use the CheckBox content tag within a Grid data source to present an interactive check box to users.

The CheckBox tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the Checkbox tag will be resolved as an interactive check box. The user can check and uncheck the box, and this state change will be submitted back to the source file.

The CheckBox tag can also be used to present an interactive toggle switch within a grid. One of the parameters of the CheckBox tag determines whether it displays as a check box (the default behavior) or as a toggle switch (optional behavior). The toggle switch behavior is only available for thematic grids.

NOTE: If you want to display a static check box symbol in a grid (meaning you do not need the user to be able to check and uncheck the box), then you can use the Symbol tag instead.

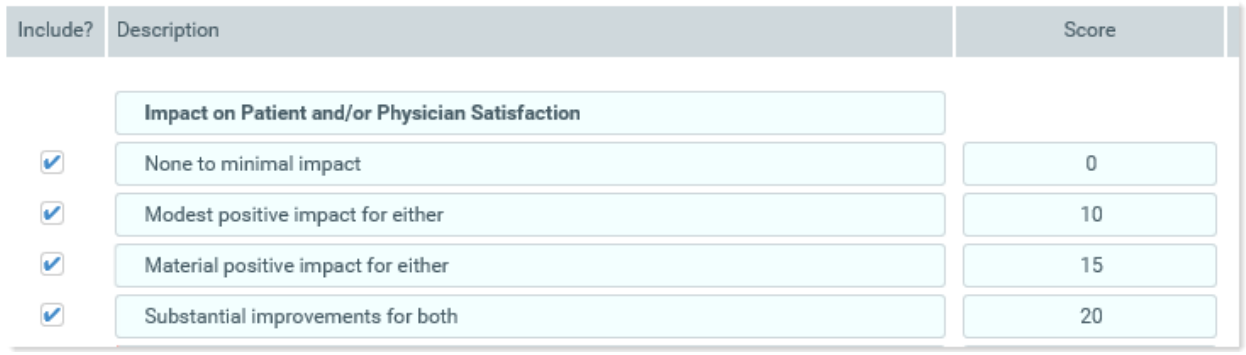


P52 [CheckBox;TargetCell=+24;]

Home | [CP18] CP Matrix

	M	O	P	Q	R
42	[Grid;Main]	[RowStyle]	[Column]	[Column]	[Column]
43	[ColumnWidth]		60px	500px	160px
44	[ColumnStyle]		colhead1,center	colhead1,left	colhead1,
45	[Fixed]	colhead,input-lin	Include?	Description	Score
49	[ColumnStyle]		col, center	col, left, bold	col, center
50	[Row]	input-lineitem		Impact on Patient and/or Physician Satisfaction	
51	[ColumnStyle]		col, center	col, left	col, center
52	[Row]	input-lineitem	[CheckBox;TargetCell=+24;]	None to minimal impact	0
53	[Row]	input-lineitem	[CheckBox;TargetCell=+24;]	Modest positive impact for either	10
54	[Row]	input-lineitem	[CheckBox;TargetCell=+24;]	Material positive impact for either	15
55	[Row]	input-lineitem	[CheckBox;TargetCell=+24;]	Substantial improvements for both	20

Example CheckBox tags in a Grid data source



Include?	Description	Score
	Impact on Patient and/or Physician Satisfaction	
☑	None to minimal impact	0
☑	Modest positive impact for either	10
☑	Material positive impact for either	15
☑	Substantial improvements for both	20

Rendering of example tags in an Axiom form

Content tag syntax for check boxes

The syntax for the CheckBox content tag is as follows:

```
[Checkbox; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; Text=LabelText; Tooltip=Text; Symbol-
checked=SymbolName; Symbol-unchecked=SymbolName; ReadOnly=True/False;
ToggleSwitch=True/False; OnText=Text; OffText=Text;
AutoSubmit=Enabled/Disabled/Grid; ColumnStyle=StyleName; Columns=Number;]
```

Parameters can be listed in any order after the CheckBox tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell to place the current state of the check box—checked (1) or unchecked (0). You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the CheckBox tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.
FormState	The key name for the value to be stored in the form state. For example, IncludeItem. The name must be unique. It is best to define a name that relates to the specific purpose of this check box. When a form state key name is defined, the current state of the check box is not stored anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Excel Client or the Windows Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

Parameter	Description
SharedVariable	The shared variable name to save the selected value as. For example, IncludeItem. When a variable name is defined, the current state of the check box is not stored anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
Text	Optional. If defined, the label text displays to the right of the check box. Otherwise, only the check box displays in the cell. You can define the text within the tag directly, or you can use a bracketed cell reference to read the text from another cell. For more information, see Referencing cells in content tag parameters. This parameter does not apply and is ignored if the ToggleSwitch parameter is set to True.
Symbol-checked	Optional. The symbol to use to represent the checked state of the check box. By default this is icon-check. You can use any symbol allowed by the Symbol content tag. When using the Data Source Assistant / Tag Editor, you must first select the **Use Custom Symbol** check box in order to expose this option. You can then click the **...** button to select a symbol from the **Choose Symbol** dialog. Alternatively, you can use a bracketed cell reference to read the symbol name from another cell. For more information, see Referencing cells in content tag parameters. This parameter does not apply and is ignored if the ToggleSwitch parameter is set to True.

Parameter	Description
Symbol-unchecked	Optional. The symbol to use to represent the unchecked state of the check box. By default this is icon-check-empty. You can use any symbol allowed by the Symbol content tag. When using the Data Source Assistant / Tag Editor, you must first select the **Use Custom Symbol** check box in order to expose this option. You can then click the **...** button to select a symbol from the **Choose Symbol** dialog. Alternatively, you can use a bracketed cell reference to read the symbol name from another cell. For more information, see Referencing cells in content tag parameters. This parameter does not apply and is ignored if the ToggleSwitch parameter is set to True.

Toggle switch parameters

If you want to display a toggle switch instead of a check box within the grid, you can use the following parameters.

Parameter	Description
ToggleSwitch	Specifies whether the interactive control displays as a check box or as a toggle switch. By default this is False, which means the control displays as a check box. If True, then the control displays as a toggle switch. When toggle switch behavior is enabled, the following parameters do *not* apply and are ignored: • Text • Symbol-checked • Symbol-unchecked When using the Data Source Assistant / Tag Editor, this option is labeled **Use Toggle Switch**. When you select the check box to use the toggle switch, the **On Text** and **Off Text** options are exposed.
OnText	Optional. Defines text to display when the switch is toggled to On. By default, the text **On** is used if no alternate text is defined.
OffText	Optional. Defines text to display when the switch is toggled to Off. By default, the text **Off** is used if no alternate text is defined.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: ToggleSwitch, OnText, OffText, AutoSubmit, Columns, ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters:
`NoAutoSubmit=`*`True/False`*`; Foreground=`*`Color`*`; Background=`*`Color`*.

For more information on using these parameters, see the topic on using the CheckBox tag in Axiom Software Help.

Behavior notes

- To set the initial state of the check box, you can enter 1 (checked) or 0 (unchecked) into the target cell. If you are using form state or a shared variable instead of a target cell, then the default value can be set by entering 1 (checked) or 0 (unchecked) into the default value parameter of the GetFormState function or the GetSharedVariable function. These functions can be located anywhere in the sheet.
- It is assumed that the target cell is off to one side (not visible in the formatted grid), within a work column. If you are saving the user's input to the database, the column to save is the column containing the target cell (not the column containing the input cell).
- In spreadsheet-formatted grids, check box cells are not recognized as editable for purposes of tab navigation.
- In spreadsheet-formatted grids, the only way to check or clear the check box is to use the mouse or equivalent (for example, tapping on a tablet surface). Keyboard gestures such as pressing the space bar will not change the check box state.

Examples

```
[Checkbox;TargetCell=K]
```

This example displays a check box using the minimum required parameters. The user can check or uncheck the box, and either 1 or 0 will be written to column K within the current row (for example if the tag is in row 22, the target cell is K22).

```
[Checkbox;TargetCell=K23;Text=Delete;NoAutoSubmit=True]
```

In this example, text has been defined to display to the right of the check box. Also, when the user changes the state of the check box, the form will not be updated automatically (meaning a submit will not occur, regardless of whether the formatted grid is set to auto-submit).

```
[Checkbox;FormState=CheckBoxState]
```

In this example, the state of the check box is stored in form state memory under the key name CheckBoxState, instead of placing the state in a target cell. When the Axiom form is used as a dialog, this value can be passed to the currently active spreadsheet file (as True or False).

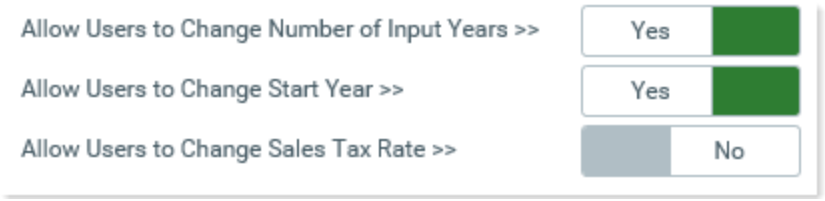
```
[Checkbox;SharedVariable=CheckBoxState]
```

In this example, the state of the check box is stored in memory in the list of variables for the shared form instance, as the value for the variable CheckBoxState. For example, you might do this so that the user can enable or disable something in a child form (displayed using the Embedded Form component), and

then that state can be referenced in the parent form as well as other child forms within the shared form instance.

```
[CheckBox;TargetCell=H;ToggleSwitch=True;OnText=Yes;OffText=No;]
```

In this example, the ToggleSwitch parameter is used so that the interactive control displays as a toggle switch instead of a check box. Additionally, the OnText and OffText parameters are used to change the On/Off text shown on the toggle switch.



Checkbox tag displayed as toggle switches

Using date pickers in Formatted Grids

You can use the DatePicker content tag within a Grid data source to allow users to select a date from a calendar control.

The DatePicker tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the DatePicker tag will be resolved as an interactive date picker. The user can select a date from a calendar control, and this date will be submitted back to the source file.

NOTE: The DatePicker tag is only supported for use in thematic grids. If a DatePicker tag is present in a spreadsheet-formatted grid, it will display as the raw tag text.

Content tag syntax for date pickers

The syntax for the DatePicker content tag is as follows:

```
[DatePicker; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; MinDate=Date; MaxDate=Date;
AutoSubmit=Enabled/Disabled/Grid; Tooltip=Text; ColumnStyle=StyleName;
Columns=Number; ReadOnly=True/False]
```

Parameters can be listed in any order after the DatePicker tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

Parameter	Description
TargetCell	The cell to place the selected date. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the DatePicker tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. The selected date is written to the target cell as an Excel date/time serial number. Keep in mind that the date picker allows users to clear the date and return a blank value. If the selected date is being used in calculations and/or to drive other components in the form, make sure to construct the relationship to accommodate a possible blank value. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.
FormState	The key name for the value to be stored in the form state. For example, StartDate. The name must be unique. It is best to define a name that relates to the specific purpose of this date picker. When a form state key name is defined, the user's selected date is not placed anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Excel Client or the Windows Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

Parameter	Description
SharedVariable	The shared variable name to save the selected value as. For example, StartDate. When a variable name is defined, the user's selected date is not placed anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
MinDate	Optional. Specify the earliest date that is valid for a user to select in the date picker. If specified, the calendar control will not allow the user to select a date that is earlier than this date. You can enter a valid date, or you can use a bracketed cell reference to read the date from the referenced cell. This approach is useful if you want to dynamically determine the date, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters. When using the Data Source Assistant / Tag Editor, this parameter is labeled as **Earliest Date**. In this environment, you can use the **Choose a date** button **[...]** to open a calendar control and select a date, or you can type a date or a bracketed cell reference. **NOTE:** If the earliest date can change using a formula, and the selected date is now invalid due to the changed earliest date, then the date picker will display as blank rather than showing the invalid date.

Parameter	Description
MaxDate	Optional. Specify the latest date that is valid for a user to enter into the date picker. If specified, the calendar control will not allow the user to select a date that is later than this date. You can enter a valid date, or you can use a bracketed cell reference to read the date from the referenced cell. This approach is useful if you want to dynamically determine the date, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters. When using the Data Source Assistant / Tag Editor, this parameter is labeled as **Latest Date**. In this environment, you can use the **Choose a date** button **[...]** to open a calendar control and select a date, or you can type a date or a bracketed cell reference. **NOTE:** If the latest date can change using a formula, and the selected date is now invalid due to the changed latest date, then the date picker will display as blank rather than showing the invalid date.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Behavior notes

- To set the initial state of the date picker, you can enter a date into the target cell. If the target cell does not have a value, the calendar control will highlight the following date by default when the user opens it:
  - If today's date falls within the min and max dates, today's date will be selected.
  - Otherwise, the earliest available date as defined by the min date will be selected.
- If you are using form state or a shared variable instead of a target cell, then the default value can be set by entering a date into the default value parameter of the GetFormState function or the GetSharedVariable function. These functions can be located anywhere in the sheet.
- It is assumed that the target cell is off to one side (not visible in the formatted grid), within a work column. If you are saving the user's input to the database, the column to save is the column containing the target cell (not the column containing the DatePicker tag).

Examples

```
[DatePicker; TargetCell=K]
```

This example displays a date picker using the minimum required parameters. The user can select a date, and that date will be written to column K within the current row (for example if the tag is in row 22, the target cell is K22).

```
[DatePicker; TargetCell=K; MinDate=1/1/2016; MaxDate=12/31/2017]
```

In this example, the MinDate and MaxDate parameters are used to restrict the dates that the user can select.

```
[DatePicker; TargetCell=K; MinDate=1/1/2016; MaxDate=12/31/2017;
ColumnStyle=auto-width]
```

In this example, the ColumnStyle parameter is being used to automatically set the width of the date picker (instead of filling the column width). This would be necessary if you want the control to auto-size but the control is in a column that is styled to fit width.

```
[DatePicker; TargetCell=K; MinDate=[M]; MaxDate=[N]; ColumnStyle=[O]]
```

This is the same as the previous example, except in this case bracketed cell references are used to read the min date, max date, and column style from the designated cells in the current row.

```
[DatePicker; FormState=StartDate; MinDate=[M]; MaxDate=[N]; ColumnStyle=[O]]
```

In this example, the selected date is being stored in form state memory rather than being placed in a target cell. When the Axiom form is used as a dialog, this date can be passed to the currently active spreadsheet file.

```
[DatePicker; SharedVariable=StartDate; MinDate=[M]; MaxDate=[N]; ColumnStyle=
[O]]
```

In this example, the selected date is being stored in memory as a shared variable rather than being placed in a target cell. For example, you might do this so that the user can select a date in a child form (displayed using the Embedded Form component), and then that date can be referenced in the parent form as well as other child forms within the shared form instance.

Using drop-down lists in Formatted Grids

You can display a drop-down list in a Formatted Grid component by setting up a cell in the Grid data source to use the Select content tag. The Select tag defines the source of the list, as well as other list options. The list can be sourced as follows:

- From a column in a table
- From an Axiom query defined in the source file
- From a ComboBox data source defined in the source file

The following topics explain how to set up a Select tag using each of these list sources. The available parameters and setup requirements vary depending on which list source is used.

NOTE: For spreadsheet-formatted grids, it is also possible to create a drop-down list using Excel's Data Validation feature. This approach has fewer options, but may be appropriate for simple lists that do not relate to data stored in Axiom Software. For more information, see Using Data Validation in spreadsheet-formatted grids.

Creating a drop-down list based on a table column

You can use the Select content tag within a Grid data source to allow users to select an item from a column in a table.

The list of items displays as a combo box, allowing selection from a drop-down list and/or typing into the box to find a specific value. Only the first 100 items display in the drop-down list, but all items can be selected by typing to search. The Select tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the Select tag will render as the combo box.

Content tag syntax for table column drop-down lists

The syntax for the Select content tag is as follows:

```
[Select; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; DisplayCell=CellAddress; ValueColumn=ColumnName;
DescriptionColumn=ColumnName; SortColumn=ColumnName; DisplayFormat=Text;
Searchable=True/False; Placeholder=Message; MultiSelect=True/False;
Filter=FilterStatement; ReadOnly=True/False; AutoSubmit=Enabled/Disabled/Grid;
Tooltip=Text; ColumnStyle=StyleName; Columns=Number;]
```

Parameters can be listed in any order after the Select tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

When using the Data Source Wizard / Tag Editor, you must specify the **List Items Source** as **Table** in order to configure the tag to use a table column. This selection is used to show the correct parameters for this configuration.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell to place the selected value. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the Select tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. If there is content in the target cell, that content will be displayed as the selected value for the combo box (unless a DisplayCell is specified). If you want to display a previously selected value that was saved to the database (such as when the form is subsequently opened and a value is queried back in), then you can use a formula in the target cell to bring in this value. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.
FormState	The key name under which the selected value will be stored in form state memory. For example, VPName. When a form state key name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Desktop Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

Parameter	Description
SharedVariable	The shared variable name to save the selected value as. For example, ProposalName. When a variable name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
ValueColumn	The column to provide the list of values for the combo box. Enter a fully-qualified Table.Column name such as `Acct.Acct`. Multi-level lookups can be used. You can specify any column from any customer-defined table in your system. System tables such as Axiom.Aliases are not supported for use with refresh variables and cannot be used. When using columns with lookups (including multi-level lookups), the final lookup table is considered the primary table. For example, if you specify `GL2018.Dept`, this is the same as specifying `GL2018.Dept.Dept`, so the Dept table is the primary table. Any columns listed in filters and as additional columns must be resolvable from the primary table, or must contain a fully qualified path from the starting table (GL2018 in this example). When using columns with lookups, the starting table impacts the list of items to be returned from the value column. For example, `GL2018.Dept` returns only the departments used in the GL2018 table, whereas `Dept.Dept` returns the full list of departments defined in the Dept table.

Parameter	Description
DescriptionColumn	Optional. The column that contains descriptions for the value column, specified using a fully qualified Table.Column name or an alias name. For example: Acct.Description. This property only applies if the value column is a key column or a validated column. By default, the primary table's first description column will be displayed in the list if no alternate description column is specified in the tag. You only need to complete the DescriptionColumn parameter if the table has more than one description column and you want to specify a different description column. However, if you are using the DisplayFormat parameter to define a custom display format, then the DescriptionColumn parameter does not apply. Instead, you should include the description column in the custom display format as desired.
Filter	Optional. A filter criteria statement to limit the values available for selection. The filter impacts both what displays in the drop-down list, and what is available when searching using the filter box. If the value column uses a lookup, then the column in the filter criteria statement must be resolvable from the primary table, or must use a fully qualified path from the starting table.
SortColumn	Optional. The column by which to sort the list of values. By default, the list is sorted by the display format if defined, and by the value column if no display format is defined. You can use this property to override the default sort and instead specify a different column to sort by. If the value column uses a lookup, then the column must be resolvable from the primary table, or must use a fully qualified path from the starting table.

Parameter	Description
DisplayCell	Optional. The cell that contains content that you want to display in the combo box other than the selected value. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. For example, due to limited space in the grid, you may want to display the value's description in the cell instead of the literal selected value. The display cell can contain the description of the selected value (as returned via a GetData function). The display cell contents only display if a value has been selected (meaning, the target cell has content). The display cell is not meant to be a substitute for the placeholder text.
DisplayFormat	Optional. Defines a display format for the items in the list, and specifies additional columns to display. By default, items in the list are displayed as: *KeyColumn - DescriptionColumn* If you want to specify a different format and/or use additional columns, then you can indicate the display format here. Use fully qualified Table.Column syntax and place column references in curly brackets. For example, you could indicate something like: {Acct.Acct} - {Acct.Description} ({Acct.Category}) This would display account items in the following format: 8000 - Facilities (Overhead) Any columns listed should use fully qualified Table.Column syntax. If the value column uses a lookup, then any additional columns must be resolvable from the primary table, or must use a fully qualified path from the starting table. Additional columns included in the display format are searchable within the list.

Parameter	Description
Placeholder	Optional. A message to display within the combo box when no value has yet been selected. This only applies when the target cell does not have any contents, or when the form state key or shared variable does not currently have an assigned value. For example: "Select a Department." Once a value has been selected, the contents of the target cell (or the stored form state / shared variable value) display instead of the placeholder text. You can define the placeholder text within the tag directly, or you can use a bracketed cell reference to read the placeholder text from another cell. For more information, see Referencing cells in content tag parameters. **NOTES:** • The appearance of the placeholder text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the placeholder text displays in a lighter color than selected values, but not always. • If multi-select is enabled, then the placeholder text is also used as the title of the multi-select dialog.
Searchable	Optional. Specifies whether the list is searchable (True/False). By default, this is True, which means the list is searchable by typing values into the combo box. If you want the list to be a static drop-down list instead, indicate False. If the list has more than 100 items, then the list must be searchable or else users will not be able to select all items in the list. Only the first 100 items are displayed in the drop-down list, but all items can be found by searching. **NOTE:** If MultiSelect is True, then this parameter does not apply. The dialog box where users can select multiple values always has a search box to filter the list. It is recommended to disable this parameter so that users cannot attempt to type inside the combo box (which will just cause the multi-select dialog to open).

Parameter	Description
MultiSelect	Optional. Specifies whether users can select multiple values in the list (True/False). By default this is False, which means that users can only select a single value. If True, then users can select multiple values in the list. When the user clicks on the combo box, a dialog opens instead of a drop-down list. The user can select check boxes for the values they want to select. When the user clicks OK to close the dialog, the selected items are written back to the target cell using a comma-separated list. If the items are from a string column, they are automatically wrapped in single quotation marks so that they can be used in a filter using an IN operator. The combo box in the form displays the same list, unless a display format cell is being used to change the display. For more information on how users interact with the multi-select dialog, see Multi-select dialog behavior. **NOTE:** If multi-select is enabled, it is recommended to define placeholder text because that text will display as the title of the multi-select dialog.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: AutoSubmit, MultiSelect, Columns, ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Select tag in Axiom Software Help.

Tag examples

```
[Select; TargetCell=K; ValueColumn=Acct.Acct; Searchable=False;
Placeholder=Select Account; Filter=Acct.Category='Revenue']
```

This example allows the user to select from a list of revenue accounts. The cell will display the text "Select Account" until an account is selected. The selected value will be placed in column K on the current row (for example if the tag is defined on row 24, then the target cell is K24). The optional Searchable parameter is set to False, so users will not be able to filter this list. There are probably only a handful of revenue accounts so a static drop-down list is appropriate.

```
[Select; FormState=Location; ValueColumn=Loc.Loc]
```

This example allows the user to select from a list of defined locations. The selected location will be stored in form state memory for the Axiom form, under the key name Location. When the Axiom form is used as a dialog or a task pane, this location value can be passed to the currently active spreadsheet file.

```
[Select; SharedVariable=Location; ValueColumn=Loc.Loc]
```

This example is the same as the previous example, except that it is using a shared variable instead of form state. The selected location will be stored in the list of variables for the shared form instance, as the value for the variable Location. For example, you might do this so that the user can select a location in a parent form, and then that location can be referenced in child forms displayed using the Embedded Form component.

```
[Select; TargetCell=K; DisplayCell=L; ValueColumn=Acct.Acct;
DescriptionColumn=Acct.FullDescription; Placeholder=Select Account]
```

In this example, the selection will be placed in column K within the current row. Additionally, the display cell parameter is used so that when a user selects an account from the list, the Select cell will display the contents of cell L24 instead of the account code placed in K24. The DescriptionColumn parameter is used to explicitly indicate a description column (perhaps in this case the table has two description columns).

```
[Select; TargetCell=B10; ValueColumn=Dept.Dept; DisplayFormat={Dept.Dept}
({Dept.Description}) - {Dept.Company}; Placeholder=Select Department]
```

In this example, a specific cell is indicated as the target cell instead of assuming the current row. The DisplayFormat parameter is used to define an alternate display for the items in the list. The additional column listed in the display format is also searchable.

```
[Select; TargetCell=K; ValueColumn=Acct.Acct; Placeholder=Select Account;
Columns=2; ColumnStyle=center]
```

In this example, the Columns parameter is used to span the combo box across two columns, and the ColumnStyle parameter is used to apply the center style to this cell, regardless of which style is currently being applied to the column. These parameters only apply to thematic grids.

```
[Select; TargetCell=K; ValueColumn=Dept.Dept; Placeholder=Select Departments;
MultiSelect=True]
```

In this example, the MultiSelect parameter is used to enable selection of multiple values. Values are displayed in a dialog so that the user can select multiple values using check boxes. The search box at the top can be used to filter the list and search for specific values.

Displaying starting values in the Select cell

The Placeholder parameter can be used to display text such as "Select a department" when no item has been selected yet. However, once the user has selected an item, and that item has been saved to the database, you likely want that item to be displayed in the cell the next time the file has opened. You can do this as follows:

- Use a formula in the target cell that returns the saved item in the database if applicable, otherwise the cell is blank.
- If the target cell contains a value, that value is displayed in the cell with the drop-down list. If the target cell does not yet contain a value, then the placeholder text is used.
- If the user selects a new value from the list, that value will overwrite the formula in the target cell. When the save-to-database is performed, that new value will replace the old value in the database. However, because the file itself is not saved, the formula is preserved the next time the file is opened, and it will now return the new value in the database.

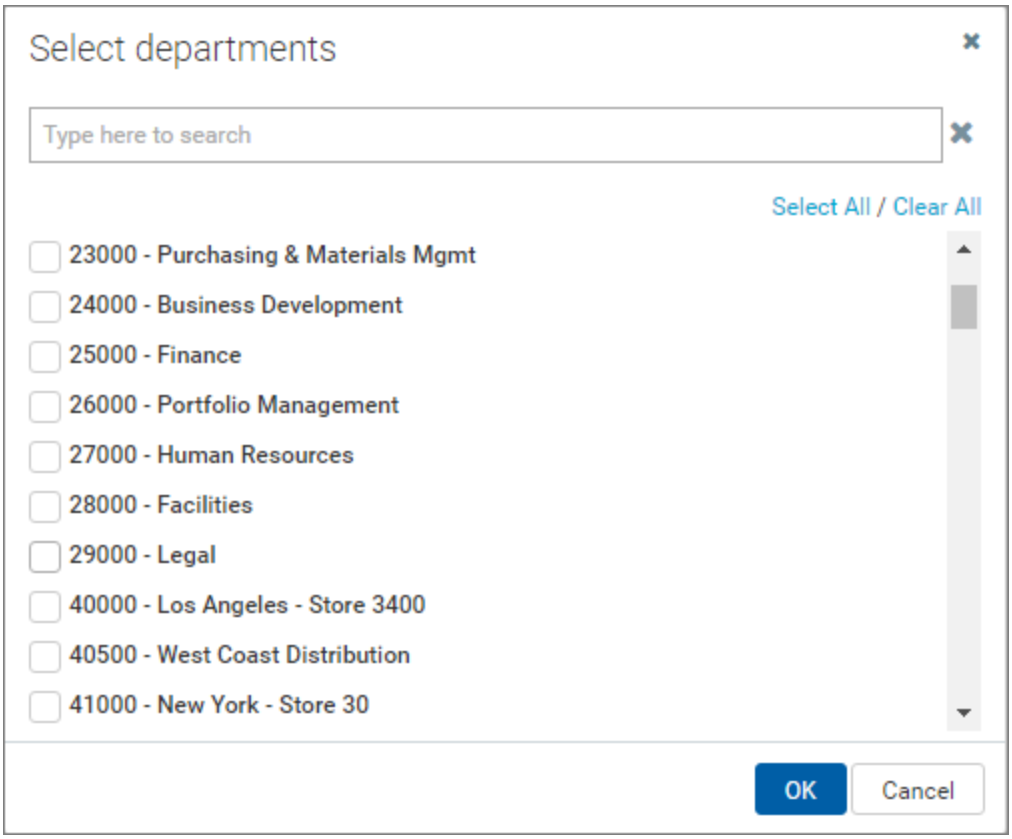
If you are using a display cell as well, then that cell should also use a formula so that it displays the desired value when the target cell contains a value, otherwise it returns blank (so that the Placeholder text can be used).

This discussion does not apply when using form state or a shared variable. In that case, you can use either the GetFormState function or the GetSharedVariable function to set a starting value, which will then display as the selected value for the list. The functions can be located anywhere in the file.

Multi-select dialog behavior

When using the multi-select dialog, users can select and clear values as follows:

- Users can use the search box in the dialog to find any value in the list. Users can select an item, then clear the search or define a different search, and select other items. All selected items will be remembered and applied when the user clicks **OK**.
- **Select All** selects all currently visible items, whereas **Clear All** clears all selected items. Select All is primarily intended to be used when the user filters the dialog and then wants to select all items resulting from the filter. Clear All is intended to provide a way to clear all selected values and start over.
- If a user selects items and clicks **OK**, then the user re-opens the multi-select dialog, all previously selected items remain selected even if they are not currently visible in the dialog. If the user selects more items without clearing any items, those items are added to the previously selected items. If the user wants to clear all items and start over, they must click **Clear All** to clear the previous selections.



Example multi-select dialog

Creating a drop-down list based on an Axiom query

You can use the Select content tag within a Grid data source to allow users to select an item from a list generated by an Axiom query.

By default, the list of items displays as a combo box, allowing selection from a drop-down list and/or typing into the box to find a specific value. Only the first 100 items in the list display in the drop-down list, but all items can be selected by typing to search. The Select tag syntax has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the Select tag will render as the combo box.

Content tag syntax for Axiom query drop-down lists

The syntax for the Select content tag is as follows:

```
[Select; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; DisplayCell=CellAddress; AQ=Sheet!AQName;
ValueColumn=ColumnName; DescriptionColumn=ColumnName; DisplayFormat=Text;
Placeholder=Message; MultiSelect=True/False; ReadOnly=True/False;
Filter=FilterStatement; AutoSubmit=Enabled/Disabled/Grid; Tooltip=Text;
ColumnStyle=StyleName; Columns=Number;]
```

Parameters can be listed in any order after the Select tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

When using the Data Source Wizard / Tag Editor, you must specify the **List Items Source** as **Axiom query** in order to configure the tag to use an Axiom query. This selection is used to show the correct parameters for this configuration.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell to place the selected value. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the Select content tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. If there is content in the target cell, that content will be displayed as the selected value for the combo box (unless a DisplayCell is specified). If you want to display a previously selected value that was saved to the database (such as when the form is subsequently opened and a value is queried back in), then you can use a formula in the target cell to bring in this value. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.

Parameter	Description
FormState	The key name under which the selected value will be stored in form state memory. For example, VPName. When a form state key name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Desktop Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).
SharedVariable	The shared variable name to save the selected value as. For example, ProposalName. When a variable name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
AQ	The name of the Axiom query to use as the source of the list. This parameter uses the following syntax: *SheetName*!*AQName*. For example: Sheet2!AQList. See the discussion below for more details on how to set up the Axiom query.
ValueColumn	Optional. The table column that contains the list of values, specified using a fully qualified Table.Column name. For example: Dept.Dept. By default, the first column in the Axiom query field definition is assumed as the value column. You only need to specify the value column in the tag if it is not the first entry in the field definition.

Parameter	Description
DescriptionColumn	Optional. The column that contains descriptions for the value column, specified using a fully qualified Table.Column name. For example: Dept.Description. By default, the second column in the Axiom query field definition is assumed as the description column. You only need to specify the description column in the tag if it is not the second entry in the field definition and if you are not using the DisplayFormat parameter to define a custom display format. However, if you are using the DisplayFormat parameter to define a custom display format, then the DescriptionColumn parameter does not apply. Instead, you should include the description column in the custom display format as desired.
Filter	Optional. A filter criteria statement to limit the values available for selection. The filter impacts both what displays in the drop-down list, and what is available when searching using the filter box. When using an Axiom query as a source, you can use the filter parameter, or you can define a filter in the Axiom query settings (or both).
DisplayCell	Optional. The cell that contains content that you want to display in the combo box other than the selected value. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. For example, due to limited space in the grid, you may want to display the value's description in the cell instead of the literal selected value. The display cell can contain the description of the selected value (as returned via a GetData function). The display cell contents only display if a value has been selected (meaning, the target cell has content). The display cell is not meant to be a substitute for the placeholder text.

Parameter	Description
DisplayFormat	Optional. Defines a display format for the items in the list, and specifies additional columns to display. By default, items in the list are displayed as: *KeyColumn - DescriptionColumn* If you want to specify a different format and/or use additional columns, then you can indicate the display format here. Use fully qualified Table.Column syntax and place column references in curly brackets. For example, you could indicate something like: {Acct.Acct} - {Acct.Description} ({Acct.Category}) This would display account items in the following format: 8000 - Facilities (Overhead) Any column used in the display format must also be included in the field definition of the Axiom query.
Placeholder	Optional. A message to display within the combo box when no value has yet been selected. This only applies when the target cell does not have any contents, or when the form state key or shared variable does not currently have an assigned value. For example: "Select a Department." Once a value has been selected, the contents of the target cell (or the stored form state / shared variable value) display instead of the placeholder text. You can define the placeholder text within the tag directly, or you can use a bracketed cell reference to read the placeholder text from another cell. For more information, see Referencing cells in content tag parameters. **NOTES:** • The appearance of the placeholder text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the placeholder text displays in a lighter color than selected values, but not always. • If multi-select is enabled, then the placeholder text is also used as the title of the multi-select dialog.

Parameter	Description
Searchable	Optional. Specifies whether the list is searchable (True/False). By default, this is True, which means the list is searchable by typing values into the combo box. If you want the list to be a static drop-down list instead, indicate False. If the list has more than 100 items, then the list must be searchable or else users will not be able to select all items in the list. Only the first 100 items are displayed in the drop-down list, but all items can be found by searching. **NOTE:** If MultiSelect is True, then this parameter does not apply. The dialog box where users can select multiple values always has a search box to filter the list. It is recommended to disable this parameter so that users cannot attempt to type inside the combo box (which will just cause the multi-select dialog to open).
MultiSelect	Optional. Specifies whether users can select multiple values in the list (True/False). By default this is False, which means that users can only select a single value. If True, then users can select multiple values in the list. When the user clicks on the combo box, a dialog opens instead of a drop-down list. The user can select check boxes for the values they want to select. When the user clicks OK to close the multi-select dialog, the selected values are written back to the target cell using a comma-separated list. If the values are from a string column, they are automatically wrapped in single quotation marks so that they can be used in a filter using an IN operator. The combo box in the form displays the same list, unless a display format cell is being used to change the display. For more information on how users interact with the multi-select dialog, see Multi-select dialog behavior. **NOTE:** If multi-select is enabled, it is recommended to define placeholder text because that text will display as the title of the multi-select dialog.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: AutoSubmit, MultiSelect, Columns, ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Select tag in Axiom Software Help.

Axiom query setup for use in a Select tag

When a user interacts with the combo box to select an item, the specified Axiom query is run *in memory only* (meaning, no values are populated within the sheet where the query is configured). The results of the query are used to populate the list.

The Axiom query should be set up as follows:

- The first column in the field definition should be the value column for the list—meaning the values to be selected. If the value column is a key column, then the second column in the field definition should be the description column for the key values.

  If for some reason you do not want these columns to be the first columns of the query, then you can use the `ValueColumn` and `DescriptionColumn` parameters of the Select tag. This explicitly tells Axiom to use these columns as the value and description, regardless of where they are located in the Axiom query field definition.

- The field definition of the query must also contain any additional columns used in the `DisplayFormat` parameter of the Select tag. These columns must be placed after the key and description columns. All columns in the field definition must be contiguous (no blank cells in between).

- It is recommended to use fully qualified Table.Column names in the field definition for the Axiom query. If you define a display format for the combo box, the display format must use fully qualified columns, and they must match the corresponding field definition entries exactly.

- The Axiom query data filter can be used to filter the list of values returned by the query. If desired, you can also (or alternatively) use the Filter parameter for the Select tag.

- All refresh behavior options for the Axiom query should be set to **Off** (such as **Refresh on file open**, **Refresh on manual refresh**, etc.), unless you also want the query to run at those times for reasons other than the drop-down list.

- No Axiom query settings that impact the display in the sheet will apply to the drop-down list. This includes spreadsheet sorting (use data sort instead), in-sheet calc method formatting or formulas, and data range filters. The only Axiom query settings read from the sheet are the field definition entries.

Tag examples

```
[Select; TargetCell=K23; AQ=Sheet2!AQList; Placeholder=Select Category]
```

This example allows the user to select from a list of categories generated by the Axiom query named "AQList" defined for Sheet2. The cell will display the text "Select Category" until a category is selected. The selected value will be placed in cell K23. Note that omitted parameters do not need to be delimited with an "empty" semicolon.

```
[Select; FormState=Category; AQ=Sheet2!AQList]
```

This example is the same as the first example, except that instead of storing the user's selection in a target cell, the selection will be stored in form state memory under the key name of Category. When the Axiom form is used as a dialog, this category value can be passed to the currently active spreadsheet file.

```
[Select; SharedVariable=Category; AQ=Sheet2!AQList]
```

This example is the same as the previous example, except that it is using a shared variable instead of form state. The selected category will be stored in the list of variables for the shared form instance, as the value for the variable Category. For example, you might do this so that the user can select a category in a parent form, and then that category can be referenced in child forms displayed using the Embedded Form component.

```
[Select; TargetCell=K; AQ=Sheet2!AcctList; Placeholder=Select Account]
```

This example displays a list of accounts, as generated by the Axiom query named "AcctList" defined for Sheet2. The first two columns of the Axiom query are Acct.Acct and Acct.Description, so that is what will display in the drop-down list. Additionally, only a column letter is specified for the target cell, so the user's selection will be placed in column K within the current row.

```
[Select; TargetCell=K; AQ=Sheet2!AcctList; Value=Acct.Acct;
Description=Acct.Description; Placeholder=Select Account]
```

This example is the same as the prior example, except that in this case we have added the Value and Description parameters to explicitly tell Axiom which columns to display in the drop-down list. These parameters were added because the first two columns in the field definition are *not* Acct and Description.

```
[Select; TargetCell=K; AQ=Sheet2!AcctList; Placeholder=Select Account;
Columns=2; ColumnStyle=center]
```

In this example, the Columns parameter is used to span the combo box across two columns, and the ColumnStyle parameter is used to apply the center style to this cell, regardless of which style is currently being applied to the column. These parameters only apply to thematic grids.

Creating a drop-down list based on a ComboBox data source

You can use the Select content tag within a Grid data source to allow users to select an item from a defined list of values. This list is created by using a ComboBox data source.

By default, the list of items displays as a combo box, allowing selection from a drop-down list and/or typing into the box to find a specific value. The Select tag syntax has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the Select tag will render as the combo box.

Content tag syntax for data source drop-down lists

The syntax for the Select content tag is as follows:

```
[Select; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; DataSourceName=Sheet!DSName; Placeholder=Message;
```

```
MultiSelect=True/False; ReadOnly=True/False; Searchable=True/False;
AutoSubmit=Enabled/Disabled/Grid; Tooltip=Text; ColumnStyle=StyleName;
Columns=Number;]
```

Parameters can be listed in any order after the Select tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

When using the Data Source Wizard / Tag Editor, you must specify the **List Items Source** as **Named Data Source** in order to configure the tag to use a ComboBox data source. This selection is used to show the correct parameters for this configuration.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell to place the selected value. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the Select tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. If there is content in the target cell, that content will be displayed as the selected value for the combo box (using its corresponding label from the data source). If you want to display a previously selected value that was saved to the database (such as when the form is subsequently opened and a value is queried back in), then you can use a formula in the target cell to bring in this value. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.

Parameter	Description
FormState	The key name under which the selected value will be stored in form state memory. For example, VPName. When a form state key name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Desktop Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).
SharedVariable	The shared variable name to save the selected value as. For example, ProposalName. When a variable name is defined, the user's selected value is not placed anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
DataSourceName	The name of the ComboBox data source to use as the source for the list. This parameter uses the following syntax: *SheetName*!*DataSourceName*. For example: Sheet2!Category. For more information on creating the data source, see Creating a ComboBox data source for the Select tag. The `[Label]` column of the data source defines the values to display in the drop-down list. The `[Value]` column of the data source defines the value to be written to the target cell (or to form state / shared variable memory) when the user makes the selection.

Parameter	Description
Placeholder	Optional. A message to display within the combo box when no value has yet been selected. This only applies when the target cell does not have any contents, or when the form state key or shared variable does not currently have an assigned value. For example: "Select a Department." Once a value has been selected, the contents of the target cell (or the stored form state / shared variable value) display instead of the placeholder text. You can define the placeholder text within the tag directly, or you can use a bracketed cell reference to read the placeholder text from another cell. For more information, see Referencing cells in content tag parameters. **NOTES:** • The appearance of the placeholder text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the placeholder text displays in a lighter color than selected values, but not always. • If multi-select is enabled, then the placeholder text is also used as the title of the multi-select dialog.
Searchable	Optional. Specifies whether the list is searchable (True/False). By default, this is True, which means the list is searchable by typing values into the combo box. If you want the list to be a static drop-down list instead, indicate False. **NOTE:** If MultiSelect is True, then this parameter does not apply. The dialog box where users can select multiple values always has a search box to filter the list. It is recommended to disable this parameter so that users cannot attempt to type inside the combo box (which will just cause the multi-select dialog to open).

Parameter	Description
MultiSelect	Optional. Specifies whether users can select multiple values in the list (True/False). By default this is False, which means that users can only select a single value. If True, then users can select multiple values in the list. When the user clicks on the combo box, a dialog opens instead of a drop-down list. The user can select check boxes for the values they want to select. When the user clicks **OK** to close the multi-select dialog, the selected values are written back to the target cell using a comma-separated list. When using a ComboBox data source, each selected value is always wrapped in single quotation marks. Although the target cell contains the comma-separated list of values, the combo box in the form displays a comma-separated list of the corresponding labels (with no quotation marks). For more information on how users interact with the multi-select dialog, see Multi-select dialog behavior. **NOTE:** If multi-select is enabled, it is recommended to define placeholder text because that text will display as the title of the multi-select dialog.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: AutoSubmit, MultiSelect, Columns, ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Select tag in Axiom Software Help.

Tag examples

```
[Select; TargetCell=K23; DataSourceName=Lists!CategoryList; Placeholder=Select Category]
```

This example allows the user to select from a list of categories defined in the CategoryList data source. The cell will display the text "Select Category" until a category is selected. The selected value will be placed in cell K23. Note that omitted parameters do not need to be delimited with an "empty" semicolon.

```
[Select; TargetCell=K; DataSourceName=Lists!CategoryList; Placeholder=Select Category]
```

This is the same as the first example, except now only a column letter is specified for the target cell, so the user's selection will be placed in column K within the current row.

```
[Select; FormState=Category; DataSourceName=Lists!CategoryList]
```

This is the same as the first example, except that instead of storing the user's selection in a target cell, the selection will be stored in the form state under the key name of Category. When the Axiom form is used as a dialog, this category value can be passed to the currently active spreadsheet file. Placeholder text is not defined in this example, because in most cases you will use a GetFormState function to define a default value for Category instead. However, the placeholder text can still be used if desired.

```
[Select; SharedVariable=Category; DataSourceName=Lists!CategoryList]
```

This example is the same as the previous example, except that it is using a shared variable instead of form state. The selected category will be stored in the list of variables for the shared form instance, as the value for the variable Category. For example, you might do this so that the user can select a category in a parent form, and then that category can be referenced in child forms displayed using the Embedded Form component.

```
[Select; TargetCell=K; DataSourceName=Lists!CategoryList; Placeholder=Select Category; Columns=2; ColumnStyle=center]
```

In this example, the Columns parameter is used to span the combo box across two columns, and the ColumnStyle parameter is used to apply the center style to this cell, regardless of which style is currently being applied to the column. These parameters only apply to thematic grids.

Creating a ComboBox data source for the Select tag

The ComboBox data source used by the Select tag is the same data source used by the Combo Box component. To create a new data source, right-click in a cell and then select **Create Axiom Form Data Source > Combo Box**.

The tags for the Combo Box data source are as follows:

Primary tag

[ComboBox;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Combo Box component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[ComboItem]

Each row flagged with this tag defines an item to display in the combo box.

Column tags

[Label]

The display name for each item in the list. Labels should be unique. If multiple rows have the same label, then the first value with that label is used.

[Value]

The corresponding value for each label. This can be the same value as the label, or a different value.

For example, in a list of colors, both the label and the value can be the text Blue. Or, the label text can be Blue while the value is a numeric color code. Separating the label from the value allows you to display "friendly" text to end users but use any value as the selected value.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows how a ComboBox data source might look in a file:

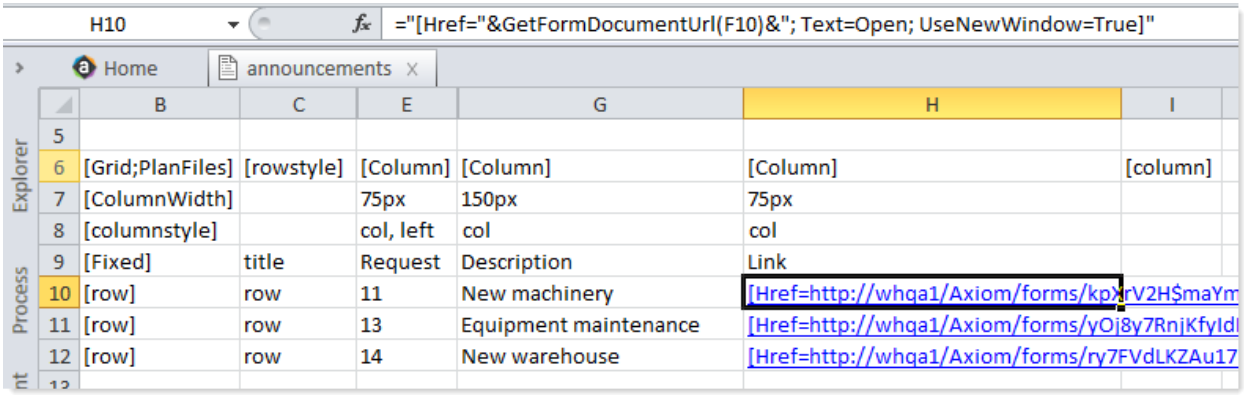
	A	B	C	D
1				
2		[ComboBox;Regions]	[Label]	[Value]
3		[ComboItem]	Consolidated	All
4		[ComboItem]	West	West
5		[ComboItem]	North	North
6		[ComboItem]	South	South
7		[ComboItem]	East	East

In this example, if the user selects the label "Consolidated" from the combo box, the value "All" will be placed in the target cell (or stored in form state / shared variable memory). The combo box displays the label of the selected value.

Using hyperlinks in Formatted Grids

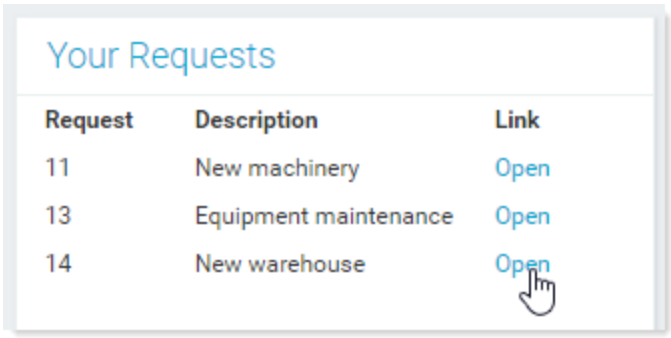
You can use the HREF content tag within a Grid data source to present hyperlinks to other files or web pages. The HREF syntax has no effect within the source file itself, but when the file is viewed as an Axiom form, the HREF tag will be resolved as a clickable hyperlink.

NOTE: Other means of creating clickable hypertext within a spreadsheet—such as using Excel's Hyperlink function, or Axiom Software's GetDocument function—will not work when the file is viewed as an Axiom form. The text will not render as clickable, and the target file or URL will not open.



	B	C	E	G	H	I
5						
6	[Grid;PlanFiles]	[rowstyle]	[Column]	[Column]	[Column]	[column]
7	[ColumnWidth]		75px	150px	75px	
8	[columnstyle]		col, left	col	col	
9	[Fixed]	title	Request	Description	Link	
10	[row]	row	11	New machinery	[Href=http://whqa1/Axiom/forms/kpXrV2H$maYm	
11	[row]	row	13	Equipment maintenance	[Href=http://whqa1/Axiom/forms/yOj8y7RnjKfyId	
12	[row]	row	14	New warehouse	[Href=http://whqa1/Axiom/forms/ry7FVdLKZAu17	

Example HREF tags in a Grid data source



Rendering of example tags in an Axiom form

Content tag syntax for hyperlinks

The syntax for the HREF content tag is as follows:

```
[Href=URL; Text=DisplayText; UseNewWindow=True/False; Tooltip=Text;
ColumnStyle=StyleName; Columns=Number;]
```

Parameters can be listed in any order. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
HREF	The URL to the target file or web page. See the following section for more information on how to generate a URL to a file within the Axiom database. When using the Data Source Assistant / Tag Editor, you can use the **[...]** button to select a file in the Axiom file system. If you browse to a file rather than specifying a URL, then the HREF parameter will contain a document reference instead of a URL. This document reference will be dynamically converted to a URL when the form is rendered. Alternatively, you can use a bracketed cell reference to read the URL from the referenced cell. This approach is useful if you want to dynamically determine the URL, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.
Text	Optional. The display text for the hyperlink. If omitted, the URL will be the display text. You can define the text within the tag directly, or you can use a bracketed cell reference to read the placeholder text from another cell. This approach is useful if you want to dynamically determine the text, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters. **NOTE:** If desired, you can use a symbol instead of display text for the hyperlink. In this case, the user can double-click the symbol to open the specified URL. To do this, replace the Text parameter with a Symbol parameter. For more information on symbol syntax, see Using symbols in Formatted Grids.
UseNewWindow	Optional. Specifies whether the URL target will open in a new browser window (True/False). If omitted, the default is False, meaning the target will open in the same browser window (replacing the current window contents). File tabs opened within the Desktop Client are treated the same as browser windows for this purpose. This parameter does not apply and should not be specified for URL targets that do not open in a browser window, such as when linking to spreadsheet Axiom files or plan file attachments. In this case, the parameter should always be omitted or set to False.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: Columns and ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Hyperlink tag in Axiom Software Help.

For example:

```
[Href=http://www.axiomepm.com;Text=Axiom Software;UseNewWindow=True]
```

This example creates a hyperlink to the Axiom Software website using the display text "Axiom Software". The hyperlink will open in a new window.

NOTE: Alternatively, you can use the functions GetFormDocumentLinkTag and GetFormResourceLinkTag to generate the HREF tag for you, given parameters such as the document ID and the display text. These functions can only be used to generate HREF tags to form-enabled documents and to plan file attachments. These functions return basic HREF syntax that cannot be modified to include optional features (like using a symbol instead display text, or defining colors in the tag).

Hyperlink behavior notes

In thematic grids, the hyperlinks generated by HREF tags are displayed in blue font with no underline by default. If desired, you can use the UNDERLINED style to apply an underline to the text.

The styles can be applied on an individual tag basis by using the ColumnStyle parameter, or at the column level by using a `[ColumnStyle]` row.

If you are using a spreadsheet-formatted grid, then the hyperlinks will display as they are formatted in the spreadsheet. Axiom Software does not apply any automatic formatting to the hyperlinks.

Generating a URL to a file in the Axiom file system

For the HREF parameter, you can type the URL directly within the content tag, or you can generate it using various Axiom functions. In most cases, unless you are pointing directly to a particular web page—such as to a page on your corporate portal—you will most likely use an Axiom function to generate a URL for a file.

The following functions can be used to generate the URL, depending on what type of file you want to open using the hyperlink:

File to Open	Function
Axiom form	Use GetFormDocumentURL to generate a URL to another Axiom form, given a file path or a document ID. Various methods are available to return the file path or the document ID of a particular file, and/or to generate a list of files and IDs. • You can use an Axiom query to the Axiom.FileSystemInfo system table to generate a list of form-enabled files of any type (reports, plan files, etc.). The query can include the full path or the document ID, either of which can be used to generate the URL. • You can use the function GetPlanFilePath to look up the file path of a particular plan file, given the plan code (for example, department 3000) and the file group name. GetPlanFilePath should be preferred over GetPlanFileDocumentID, for performance reasons. • You can look up the document ID or file path of an individual file within Axiom Explorer and place it within the function. Generally this would only be useful to create a link to a static file that all viewers of the form need access to, such as to a form-enabled report. If you want to use the hyperlink to generate a PDF copy for printing, then you can append the text `/pdf` to the URL. For more information, see Configuring an Axiom form for printing. When using GetFormDocumentURL, you can optionally apply a sheet filter to the target form, and/or optionally pass variable values to the target form.
Plan file attachment	Use GetFormResourceURL to generate a URL to a plan file attachment, given a file path or a document ID. The document IDs for plan file attachments can be returned by querying the Axiom.PlanFileAttachments table. File paths can be returned by querying the Axiom.FileSystemInfo table.
Other Axiom file	Use GetDocumentHyperlink to generate a URL to an Axiom file, given a file path or a document ID. URLs generated using GetDocumentHyperlink will open the specified file within the Desktop Client. Therefore linking to regular Axiom files from within an Axiom form should only be done when the form is intended to be viewed within the Desktop Client (or in a browser on client machines where the users have access to the Desktop Client). You can look up the ID or file path of any individual file in Axiom Explorer, or you can query the Axiom.FileSystemInfo table to get a list of files, paths, and IDs.

NOTE: When using the Tag Editor to create the HREF tag, you can browse to any file within the Axiom file system. This places a document reference within the tag, which will be dynamically converted to a URL when the form is rendered. This approach may be useful to hyperlink to a specific, static file in the Axiom file system. However, this approach does have some limitations—for example, there is no way to append `/pdf` to the generated URL to create a PDF of a form.

The following examples use a formula to generate the URL for the hyperlink:

```
="[Href="&GetFormResourceURL(G23)&";Text=Open Attachment"&H23&"]"
```

This example uses GetFormResourceURL to generate a URL to a plan file attachment, where the document ID or file path is located in cell G23. The display text incorporates the name of the attachment from cell H23. The UseNewWindow parameter is omitted because it is inapplicable to plan file attachments, which do not open in browser windows.

```
="[Href="&GetFormDocumentURL(G23)&";Text=Open Plan File for "&H23&";
UseNewWindow=True]"
```

This example uses GetFormDocumentURL to generate a URL to a form-enabled plan file, where the document ID or file path is located in cell G23. The display text incorporates the name of the associated plan code from cell H23 (something like "Dept 42000"). The plan file form will open in a new window.

Using sparklines in Formatted Grids

You can use the Sparkline content tag within a Grid data source to display a sparkline chart. Sparkline charts are minimal charts that are intended to be placed in context with related data, to display trends at-a-glance. In formatted grids, sparklines can be either traditional sparkline charts (using lines, columns, etc.) or bullet-style sparkline charts.

The Sparkline tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the Sparkline tag will be resolved as a sparkline chart.

Key Performance Indicators		Current Month Prior Month		
	Trend	November	October	Year to Date
Yield on Earning Assets		3.81%	4.06%	4.28%
Cost of Paying Liabilities		1.51%	1.59%	1.64%
Net Interest Spread		2.30%	2.30%	2.64%
Net Interest Margin		3.22%	3.43%	3.57%
Return on Assets		0.94%	0.84%	0.72%
Return on Equity		9.64%	8.55%	7.53%
Efficiency Ratio		79.66%	87.98%	87.60%
Capital Ratio		9.72%	9.85%	9.50%
Leverage Ratio		10.29	10.16	10.53

Example formatted grid using Sparkline tags

NOTE: Your Axiom Software license determines whether you have access to chart components. For more information, see Licensing requirements for Axiom forms.

Content tag syntax for sparklines

The syntax for the Sparkline content tag is as follows:

```
[Sparkline; DataSourceName=Sheet!DSName; SeriesName=Name; Href=URL;
UseNewWindow=True/False; Background=Color; ColumnStyle=StyleName;
Columns=Number]
```

Parameters can be listed in any order after the Sparkline tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

Parameter	Description
DataSourceName	The name of the data source to provide the data for the chart. This parameter uses the following syntax: *SheetName*!*DataSourceName*. For example: Sheet2!Executive. The specified data source must be either an XYChart data source (for regular sparkline charts) or a BulletChart data source (for bullet-style sparkline charts). See the following sections for more information on defining a data source for the sparkline chart.
SeriesName	The name of a series in the data source, to display that series in the chart. Typically sparkline charts display one series at a time. If omitted, then all series will display in the chart. Keep in mind that this may render the chart effectively illegible if there are more than a couple of series in the data source. **NOTE:** This parameter is required if the data source is a BulletChart data source. Only one series can display within a bullet-style sparkline.

Parameter	Description
Href	Optional. A URL to open when a user clicks on the chart in the Axiom form. For example, you can use this to open another Axiom form that displays the detailed data for the sparkline chart. For more information on options to open another Axiom form via a URL, see Generating a URL to an Axiom file or an Axiom form. Alternatively, you can use a bracketed cell reference to read the URL from the referenced cell. This approach is useful if you want to dynamically determine the URL, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.
UseNewWindow	Optional. Specifies whether the URL target will open in a new browser window (True/False). If omitted, the default is False, meaning the target will open in the same browser window (replacing the current window contents). File tabs opened within the Desktop Client are treated the same as browser windows for this purpose. This parameter does not apply and should not be specified for URL targets that do not open in a browser window, such as when linking to spreadsheet Axiom files or plan file attachments. In this case, the parameter should always be omitted or set to False.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: Columns and ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Sparkline tag in Axiom Software Help.

Data source tags (regular sparkline charts)

Sparkline tags require a data source to be defined in the source file, to define the data to display in the chart. If you want to create a regular sparkline chart, you must use an XYChart data source. This is the same data source used by Line Charts, Column Charts, etc.

Note the following exceptions when using an XYChart data source for a sparkline chart:

- Sparkline charts do not display axis labels or legends, so the `[XValueName]` row and the `[VisibleinLegend]` column are not applicable. The `[SeriesName]` column must still be used to identify series in the data source, but the series names do not display in the chart.
- Sparkline charts only support one Y-axis scale, so the `[Axis]` column is not applicable. If `[Axis]` is present in the data source and any series are assigned to the Secondary axis, those series will not display in the sparkline chart.

The tags for the data source are as follows:

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Sparkline tag. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each sparkline chart uses a single series in the data source.

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. The name identifies this series so that it can be assigned to a Sparkline tag.

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`.

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[LineStyle]

Optional. Specifies the style of the line as one of the following. If omitted, the Normal style is used. Only applies to Line series.

- **None**: No line is displayed; only markers are shown to represent the data points. `[ShowMarkers]` must be enabled or else the series will not display at all. This option is primarily intended for use in combination charts—for example, multiple bar series combined with a marker-only line series.
- **Normal**: A straight line is drawn from point to point.
- **Smooth**: A curved line is drawn from point to point.
- **Step**: The line "steps" from one point to another. The lines between points are flat, with a vertical line up or down to indicate the differential at each point.

[DashType]

Optional. Specifies the type of dash as one of the following. If omitted, the Solid style is used. Only applies to Line series.

- **Dash**: The line is drawn in dashes. The length of the dash is fixed and cannot be configured.
- **DashDot**: The line is drawn as a dash-dot-dash repeating series.
- **Dot**: The line is drawn in dots. The size of the dot is fixed and cannot be configured.
- **Solid**: The line is drawn as a solid line.

[PlotNullValues]

Optional. Specifies whether null values are plotted on the line (True/False). Only applies to Line series.

If omitted or False, then null values will result in a gap between line segments. If True, then the missing value will be interpolated, so that the line will continue from the last plotted point to the next plotted point.

NOTES:

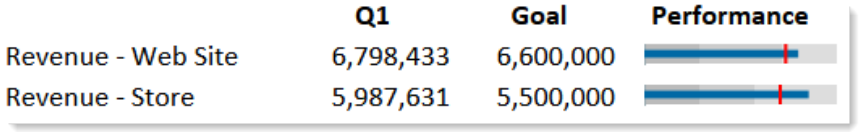
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source**. If the data already exists in the sheet, you can first highlight the labels and the values, and then use the wizard. When using the wizard, there is no separate option for Sparkline—instead you should select Line Chart or Column Chart, etc., based on what type of chart you want to display in the sparkline chart. Traditionally sparklines are line charts, but you can use any of the chart kinds supported by the XYChart data source.

To see an example data source, see the topic for the type of chart that you want to display as a sparkline. For example, if you want to display a Line Chart as a sparkline, see the topic for *Line Chart component*.

Data source tags (bullet-style sparkline charts)

Sparkline tags require a data source to be defined in the source file, to define the data to be displayed in the chart. If you want to create a bullet-style sparkline chart, then you must use a BulletChart data source.



	Q1	Goal	Performance
Revenue - Web Site	6,798,433	6,600,000	
Revenue - Store	5,987,631	5,500,000	

Example bullet-style sparkline charts

The tags for the data source are as follows:

Tag Type	Tag Syntax
Primary tag	`[BulletChart;DataSourceName]` Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save. The placement of this primary tag defines the control column and the control row for the data source. • All column tags must be placed in this row, to the right of the tag. • All row tags must be placed in this column, below the tag.
Row tags	`[Series]` Each series of data in the data source must be marked with a `[Series]` tag. A bullet chart can only display one series, but the data source can have multiple series. When you configure the component properties or the content tag, you must specify which series to use for that chart.
Column tags (required)	`[SeriesName]` This column contains the name of each series in the chart. Each bullet chart can only display one series, but the data source can have multiple series. When you configure the content tag, you must specify which series to use

Tag Type	Tag Syntax

	for that chart.
	`[Value]`
	The current value for the series.
	`[TargetValue]`
	The target value for the series.
	`[Minimum]`
	This column indicates the minimum value of the bullet chart. If this tag is omitted, the minimum value of the chart is 0. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.
	`[Maximum]`
	This column indicates the maximum value of the bullet chart. If this tag is omitted, the maximum value of the chart is 100. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

Column tags (optional)	BulletChart data sources can also use the following optional columns. These tags are not added by the data source wizard; you must manually add them to the data source if you need them.
	`[TargetColor]`
	This optional column indicates the color assignment for the target line on the bullet chart. If omitted, then colors will be dynamically determined based on the style, theme, or skin (in that order).
	You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua).
	`[BarColor]`
	This optional column indicates the color assignment for the bar on the bullet chart. If omitted, then colors will be dynamically determined based on the style, theme, or skin (in that order).
	You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua).

Column tags (ranges)	BulletChart data sources can have up to 10 ranges. Typically bullet charts have 3 value ranges, indicating whether a value is considered poor, satisfactory, or good. If no ranges are defined, then no colored ranges will display on the chart. Range tags are defined as follows (use a number from 1 to 10 in the range tag

Tag Type	Tag Syntax

	as appropriate):
	`[Range1Start]`
	The start value of the range.
	`[Range1End]`
	The end value of the range.
	`[Range1Color]`
	The color of the range. You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua). If no color is defined, then shades of gray are used to differentiate each range.
	This tag is not added by the data source wizard; you must manually add it to the data source if you need it.
	NOTE: If you want the ranges to be continuous, then the end value of one range and the start value of the next range should be the same number—for example, if range 1 is from 0 to 20, then the start value for range 2 should be 20.

Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows simple bullet chart data flagged in a sheet. In real implementations this data might be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J	K
4											
5		[BulletChart;BulletChartSource1]	[SeriesName]	[Value]	[TargetValue]	[Minimum]	[Maximum]	[Range1Start]	[Range1End]	[Range2Start]	[Range2End]
6		[Series]	ThisYear	6,798	6,600	4000	7500	4000	5000	5000	6500
7		[Series]	LastYear	5,988	5,500	3000	6500	3000	4000	4000	5000

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Bullet Chart**. If the data already exists in the sheet, you can first highlight the data (in the example above, you would highlight C6:K7) and then use the wizard. Axiom Software will add the tags as displayed in the example above. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

If adding the tags to existing values, the wizard assumes that the first column in the highlighted range holds series names, the second column holds values, and the third column holds target values. Any additional columns highlighted will be assigned as start / end range columns.

Behavior notes

- Tooltips do not display on sparkline charts within formatted grids.
- The size of the sparkline chart is dependent on the row height and column width of the cell where the Sparkline tag is placed. How the row height and column width are determined varies depending on the type of Formatted Grid (spreadsheet-formatted or thematic).

Sparkline tag examples

```
[Sparkline;DataSourceName=Data!Executive;SeriesName=Yield]
```

This example will render a sparkline chart using the Yield series in the Executive data source. The Executive data source is an XYChart data source. The type of chart (i.e. line, column, etc.) and chart color will be determined by the data source.

```
[Sparkline;DataSourceName=Data!Executive;Href=document://\Axiom\SystemFolderName_ReportsLibrary\Forms\hierarchy.xlsx?ds=True;UseNewWindow=True]
```

This example is the same as the previous example, except a URL has been specified to open when a user clicks on the sparkline chart. In this case the Tag Editor was used to browse to a form-enabled document in the Axiom Reports Library; this will be interpreted as a URL in the Axiom form.

```
[Sparkline;DataSourceName=Data!Executive]
```

This example will render a sparkline chart using all series in the Executive data source.

Using symbols in Formatted Grids

You can use the Symbol content tag within a Grid data source to display symbols in the grid. The Symbol tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the tag will be resolved as a symbol.

This is the only supported method of displaying a symbol across all platforms. In spreadsheet-formatted grids, the Wingdings font should not be used because it will not work on all platforms.

Content tag syntax for symbols

The syntax for the Symbol content tag is as follows:

```
[Symbol=SymbolName; Href=URL; UseNewWindow=True/False; Tooltip=Text;
ColumnStyle=StyleName; Columns=Number;]
```

Parameters can be listed in any order. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
Symbol	The name of the symbol to display. The name must be a valid name from the symbol library. For more information on how to browse symbols and find specific symbol names, see the following section. Alternatively, you can use a bracketed cell reference to read the symbol name from the referenced cell. This approach is useful if you want to dynamically determine the symbol, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters. If you use a cell reference, and the referenced cell currently contains a valid symbol name, then that symbol will be shown in the dialog.
HREF	Optional. The URL to a target file or web page. This can be used to display symbols that open a file or a web page when clicked. For more information on how to generate a URL to a file within the Axiom database, see the discussion in Using hyperlinks in Formatted Grids. Alternatively, you can use a bracketed cell reference to read the URL from the referenced cell. This approach is useful if you want to dynamically determine the URL, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.
UseNewWindow	Optional. Specifies whether the URL target will open in a new browser window (True/False). If omitted, the default is False, meaning the target will open in the same browser window (replacing the current window contents). File tabs opened within the Desktop Client are treated the same as browser windows for this purpose. This parameter does not apply and should not be specified for URL targets that do not open in a browser window, such as when linking to spreadsheet Axiom files or plan file attachments. In this case, the parameter should always be omitted or set to False.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: Columns and ColumnStyle.

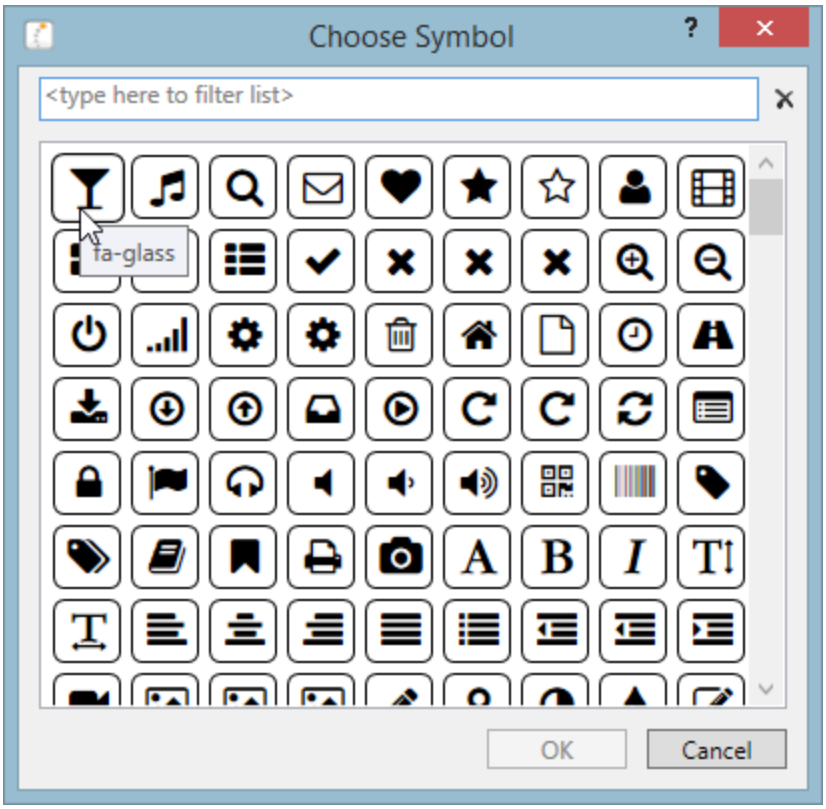
Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Symbol tag in Axiom Software Help.

Browsing available symbols

When using the Data Source Assistant / Tag Editor, you can browse the available symbols and have Axiom Software automatically create the tag for you with the appropriate symbol name.

For the **Symbol** parameter, click the Browse button **[...]** to open the **Choose Symbol** dialog. Within this dialog, you can scroll through the available symbols, or you can use the filter box at the top to find symbols based on symbol names. For example, you can type `file` to see all of the symbols that have the word "file" in the name, or you can type `fa-file-o` to find the symbol with that specific name. To find out the name for a particular symbol, hover your cursor over the symbol to see the symbol name in a tooltip.



When you have found the symbol that you want to use, select it and then click **OK** to add it to the Symbol tag.

NOTE: If you are using a cell reference for the Symbol field, then you should note the desired symbol name and then click **Cancel** to exit the dialog. If you click OK, the selected symbol name will overwrite the cell reference. After finding the symbol name, you can then go to the referenced cell and manually place the symbol name into that cell.

Tag examples

```
[Symbol=fa-filter]
```

This example displays the filter icon within the grid, when the file is viewed as an Axiom form.

```
[Symbol=fa-exclamation;foreground=red]
```

This example displays an exclamation point icon in red. You could wrap this tag in a conditional formula so that the red icon only shows when a value exceeds a certain threshold. This example is only for use in spreadsheet-formatted grids.

```
[Symbol=fa-exclamation;ColumnStyle=[N];]
```

This example serves the same purpose as the prior example, except it is for use in thematic grids. The column style is read from column N in the row that contains the tag. The cell in column N could contain a formula that changes the style based on some criteria (for example, to show the symbol in red if the value is exceeds a certain threshold).

```
="[Symbol=fa-bar-chart;Href="&GetFormDocumentURL(G23)&";UseNewWindow=True]"
```

In this example the HREF parameter is used so that the user can double-click the bar chart icon to open the specified Axiom form.

Using text boxes in Formatted Grids

You can configure cells in a Grid data source to behave as text boxes, so that the user can enter or edit text that gets submitted back to the form source file. There are two ways to do this:

- You can configure a cell in the Grid data source as unlocked. For more information on this option, see Interactivity options for Formatted Grids.
- You can use the TextArea content tag for formatted grids to configure a cell in the Grid data source as a text box. The TextArea tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the TextArea tag will render as a text box. This topic discusses how to use the TextArea tag.

The TextArea tag is the preferred approach in many cases, because it provides additional options to control the display and behavior of the editable cell. Generally speaking, simple unlocked cells should only be used for very basic edits where the additional options are not needed—for example, if you are displaying a grid of numbers in a report-style format, but users also need the option to edit the numbers.

Content tag syntax for text boxes

The syntax for the TextArea content tag is as follows:

```
[TextArea; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; Placeholder=Text; MaxLength=Number;
ReadOnly=True/False; Kind=KindName; MultiLine=True/False;
RichTextEditor=True/False; ShowToolBar=True/False; ToolBarOptions=Code;
Mask=Format; PreserveMask=True/False; MinRange=Number; MaxRange=Number;
AutoSubmit=Enabled/Disabled/Grid; Tooltip=Text; Columns=Number;
ColumnStyle=StyleName;]
```

Parameters can be listed in any order after the TextArea tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell to place the user input. You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. The target cell cannot be the same cell that contains the TextArea tag. The target cell can be anywhere in the spreadsheet and does not need to be visible within the formatted grid. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.

Parameter	Description
FormState	The key name for the text input to be stored in form state memory. For example, Description. When a form state key name is defined, the user's text input is not placed anywhere in the source file. Instead, it is stored in form state memory for the current file. If you need to reference the value within the form, you can use the GetFormState function to return the value into a cell. The FormState parameter should only be used if the form is intended to be used as a dialog in the Excel Client or the Windows Client, and you need to be able to pass values from the form to the currently active spreadsheet. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).
SharedVariable	The shared variable name to save the selected value as. For example, Description. When a variable name is defined, the user's text input is not placed anywhere in the source file. Instead, it is saved to the variable list that is stored in memory for the shared form instance. If you need to reference the value within the form, you can use the GetSharedVariable function to return the value into a cell. The SharedVariable parameter should only be used if the form is intended to be used in a composite form context (as either the parent form or a child form), and you need to share this value with other forms in the shared form instance. For more information, see Sharing variables between parent and child forms.
Placeholder	Optional. A message to display within the cell when no user input has yet been entered. This applies when the target cell does not have any contents, or when the form state key or shared variable does not currently have an assigned value. For example: "Enter a comment." Once the user has input text, the contents of the target cell (or the stored form state / shared variable value) display instead of the placeholder text. You can define the placeholder text within the tag directly, or you can use a bracketed cell reference to read the placeholder text from another cell. For more information, see Referencing cells in content tag parameters. **NOTE:** The appearance of the placeholder text depends on the skin assigned to the form, and on the browser used to view the form. In most environments the placeholder text displays in a lighter color than text values, but not always.
MaxLength	Optional. The maximum length, in characters, of the text input. Note that not all browsers will enforce this limit.

Parameter	Description
ReadOnly	Optional. Specifies whether the input cell is "active" (True/False). The default value, False, means that the cell is active and that the user can input text as needed. If True, then the cell becomes "frozen" and no further edits can be made. The cell will display the current value of the text box. This parameter can be used to control whether a user can edit the cell's value. Generally speaking, this parameter would only be used within a formula to dynamically enable / disable the text box.
Kind	Optional. Specify a kind for the text box: • `Text` (default): The text box accepts text inputs using any characters, and can use text-related options such as rich text and multi-line input. See Text kind parameters. • `InputMask`: The text box uses a defined input mask format to restrict the user input to certain character types and formats. See InputMask kind parameters. • `Numeric`: The text box only accepts number characters within an optional range. Users cannot input letters or other special characters. Numeric kind parameters. **NOTE:** When using the Data Source Assistant / Tag Editor, this option displays using the label **Type**.

Text kind parameters

The following parameters only apply if the Kind parameter is set to Text.

Parameter	Description
MultiLine	Optional. Specifies whether the text input supports multi-line input (True/False). By default, this is True. If False, then wrapped text will not be enabled for the input cell or the target cell. If RichTextEditor is enabled for the text box, then the MultiLine parameter does not apply. Rich text boxes always support multi-line input.

Parameter	Description
RichTextEditor	Optional. Specifies whether the text box supports rich text (True/False). By default this is False, which means that text entered into the text box is plain text. If True, then users can apply limited formatting to the text, such as bold or italics. This can be done by using the formatting toolbar, or by using standard shortcut keys (such as CTRL+B for bold). The text written back to the target cell uses HTML tags to indicate the formatted elements. This works the same way as the separate Text Box component with rich text enabled. For an example, see Rich text box behavior. **NOTES:** • The row height must be at least 100px in order to display the rich text box in a grid. • When using the Data Source Assistant / Tag Editor, this option displays using the label **Rich Text**.
ShowToolBar	Optional. Specifies whether the formatting toolbar displays on the text box (True/False). Only applies if RichTextEditor is True. By default this is False, which means that the text box displays as a bordered box, with no toolbar. In this case, users must use shortcut keys to format the text. If True, then the text box displays in a framed box, with a toolbar at the top to apply formatting. You can specify which formatting options display on the toolbar. The appearance is the same as the separate Text Box component with rich text enabled. For an example, see Rich text box behavior.

Parameter	Description
ToolbarOptions	Optional. Specifies which formatting options display on the toolbar, when the toolbar is enabled. By default, the toolbar contains font formatting options. It is only necessary to configure this parameter if you want the toolbar to display different options. The ToolbarOptions parameter resolves to a code that tells Axiom Software which options to display. It is recommended to always use the Tag Editor or Data Source Assistant to configure the options, to ensure that a valid code is used. To configure the options, click the **[...]** button and then select the check boxes for the desired options: • **Font options**: Enables toolbar buttons for bold, italics, and underline. This option is selected by default. If you disable this option, users can still apply font formatting by using shortcut keys (such as CTRL+B for bold). • **Alignment options**: Enables toolbar buttons for right, left, center, and justified text alignment. • **List options**: Enables toolbar buttons for numbered lists, bullet lists, and indent.

InputMask kind parameters

The following parameters only apply if the Kind parameter is set to InputMask.

Parameter	Description
Mask	Defines the input mask format for the text box. For example, you might enter `(000) 000-0000` to specify a phone number format. The input mask format uses the same rules as the separate Text Box component. For more information, see Defining the input mask format. **NOTE:** When using the Data Source Assistant / Tag Editor, this option displays using the label **Format**.

Parameter	Description
PreserveMask	Optional. Specifies whether the text submitted back to the source file includes the input mask format or not. By default this is False, which means that the user input is written back as raw text, without any formatting. If True, then the user input is written back using the same format as it is displayed to the end user. For example, imagine the input mask is defined as (000) 000-000, and the user enters (123) 456-7899. If this option is disabled, then the input will be written to the source file as raw text: 1234567899. If this option is enabled, then the input will be written to the source file as it displays to the user, including the formatting characters. You should enable this option if you need to display the input in a different component using the same format, or if you need to save the input to the database using the format.

Numeric kind parameters

The following parameters only apply if the Kind parameter is set to Numeric.

Parameter	Description
MinRange	Optional. The minimum number that can be entered into the text box, to restrict the user input to a specified numeric range. By default, this parameter is omitted, which means there is no minimum value. Specify a number if you want to define a minimum value. For example, if the minimum is set to 1, then the user can enter any number that is 1 or higher (up to the defined maximum number). If the user enters a number lower than the minimum, such as 0 or -5, then a red validation bar will display on the right side of the text box. Additionally, the tooltip will display a validation message, such as "The value must be greater than or equal to 1."
MaxRange	Optional. The maximum number that can be entered into the text box, to restrict the user input to a specified numeric range. By default, this parameter is omitted, which means there is no maximum value. Specify a number if you want to define a maximum value. For example, if the maximum is set to 100, then the user can enter any number that is 100 or lower (down to the defined minimum number). If the user enters a number higher than the maximum, such as 150, then a red validation bar will display on the right side of the text box. Additionally, the tooltip will display a validation message, such as "The value must be less than or equal to 100."

When using a numeric text box, the target cell should be formatted as Number, Currency, or Percentage, so that the inputted value displays in the desired numeric format. If you are using a shared variable or a form state key (and therefore there is no target cell), then you should format the cell containing the tag with the desired numeric format. This behavior is the same as when using a numeric Text Box component. For more information, see Using numeric text boxes.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: AutoSubmit, Columns, ColumnStyle, Kind, RichTextEditor, ShowToolbar, Mask, PreserveMask, MinRange, MaxRange. The cell only supports plain text inputs.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=`*`Color`*`; Background=`*`Color`*.

For more information on using these parameters, see the topic on using the TextArea tag in Axiom Software Help.

Behavior notes

The cell containing the TextArea tag becomes a text box when the form is rendered. The basic appearance and behavior of the text box is the same as the separate Text Box component.

Alternatively, some themes provide a special column style that you can use to display the cell as a regular cell until the user clicks in it, at which point it will display as a text box (the `click-to-edit` style). This style requires the form to use the Web Client container, and does not apply if the text box is a rich text box.

It is assumed that the target cell is off to one side (not visible in the formatted grid), within a work column. If you are saving the user's input to the database, the column to save is the column containing the target cell (not the column containing the input cell).

Spreadsheet-formatted grids

The following behavior notes apply to legacy spreadsheet-formatted grids:

- In spreadsheet-formatted grids, the cell renders as a regular editable cell, not as a text box. You should format the spreadsheet as appropriate to indicate to users that the cell is editable—for example, you may want to display the cell with a border and a background color.
- The editable cell inherits the defined formatting in the spreadsheet, with one exception: the vertical alignment of the cell is always set to top, regardless of the cell formatting.

Tag examples

```
[TextArea; TargetCell=K23; Kind=Text; Columns=4; Placeholder=Enter Comment]
```

This example allows the user to enter a comment. The input cell spans four columns wide. The cell will display the text "Enter Comment" until the user enters some text. The inputted text will be placed in cell K23.

```
="[TextArea; TargetCell=K; Kind=Text; Columns=4; Placeholder=Enter comment;
ReadOnly="&A2&"]"
```

This example is similar to the first example, except:

- The target cell is indicated by just the column letter K. The user input will be placed in column K within the current row (for example, if the tag is defined on row 22, the target cell is K22).
- The ReadOnly parameter is used to toggle the input cell on or off based on the contents of cell A2. At some point in the process, A2 could be changed to True, which would prevent users from editing the existing comment.

```
[TextArea; TargetCell=K; Kind=Text; MultiLine=False; Placeholder=Enter Name]
```

In this example, the multi-line parameter is used to turn off the wrapped text for the input. Although you could simply use an unlocked normal cell for this input, using the TextArea tag provides the following benefits:

- Separate placeholder text that does not need to be entered directly in the input cell (as it would if using a normal cell).
- The ability to place the user input into a target cell.

```
[TextArea; FormState=Filter; Kind=Text; Placeholder=Enter a Filter]
```

In this example, the text input is being stored in form state memory rather than being placed in a target cell. When the Axiom form is used as a dialog, this text input can be passed to the currently active spreadsheet file. For example, the user can define a filter statement to apply to a query in the active spreadsheet file.

```
[TextArea; SharedVariable=Description; Kind=Text; Placeholder=Enter a
Description]
```

In this example, the text input is being stored in memory as a shared variable rather than being placed in a target cell. For example, you might do this so that the user can define a description in a child form (displayed using the Embedded Form component), and then that description can be referenced in the parent form as well as other child forms within the shared form instance.

```
[TextArea; TargetCell=K; Kind=Text; MultiLine=False; ColumnStyle=col, click-
to-edit]
```

In this example, the ColumnStyle parameter is being used to apply the special click-to-edit style to the text box. This means that the cell contents will display as a regular cell until the user clicks inside the cell, at which point the normal text box borders and styling apply.

```
[TextArea; TargetCell=K; Kind=InputMask; Mask=(000) 000-0000;
PreserveMask=True;]
```

In this example, the text box uses an input mask to enforce entry of a phone number in the desired format. The input mask format is preserved so that the user input will be written to the source file using the specified format instead of as raw text.

```
[TextArea; TargetCell=K; Kind=Numeric; MinRange=1; MaxRange=100;]
```

In this example, the text box uses a numeric format and a specified number range. If the user enters a number outside of this range, say 101, then a red validation bar will display on the side of the text box and the tooltip will display a validation message.

```
[TextArea;TargetCell=+1;RichTextEditor=True;ShowToolBar=True;ToolBarOptions=2
047;Kind=Text;]
```

In this example, the text box uses rich text, and additional toolbar options of alignment and lists have been enabled. If you are using the default behavior of only showing the font formatting options on the toolbar, then the ToolbarOptions parameter is omitted.

Applying formatting to cell contents in Formatted Grids

You can use the Format content tag within a Grid data source to apply special formatting to a cell in the grid. The Format tag has no effect within the source file itself, but when the file is viewed as an Axiom form, the cell containing the Format tag will be resolved as a formatted cell that displays the contents of the target cell.

By default, the formatting of a cell is controlled by either the format of the spreadsheet (spreadsheet-formatted grids) or the specified row and column styles (thematic grids). Using the Format content tag, you can override certain formatting elements and specify them within a cell-specific tag. For example, you can:

- Span content across multiple columns
- Override the column style for the current cell
- Override foreground and background colors for the current cell

Content tag syntax for formatted cells

The syntax for the Format content tag is as follows:

```
[Format; TargetCell=CellAddress; FormState=KeyName;
SharedVariable=VariableName; Tooltip=Text; Columns=Number;
ColumnStyle=StyleName; Foreground=Color; Background=Color]
```

Parameters can be listed in any order after the Format tag. Optional parameters can be omitted.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

NOTE: These parameters assume that you are using a thematic formatted grid. If you are using a legacy spreadsheet-formatted grid, some parameters may not be available.

Parameter	Description
TargetCell	The cell that contains the content to be displayed in the current cell (the cell with the Format tag). You can specify the cell using one of the following options: • A full cell reference such as C22 or Report!C22 • A column letter such as C (where the row is the current row) • A relative column location such as +3 or -3 from the current cell For more information, see Referencing cells in content tag parameters. **NOTES:** • You must choose either TargetCell, FormState, or SharedVariable as the target of the tag. TargetCell is the default behavior and should be used unless the form is being specially designed for use with form state or shared variables. • When using the Data Source Assistant / Tag Editor, you select either **Cell**, **Form State**, or **Shared Variable** as the **Target** and then complete the field as appropriate. Your selection will be automatically rendered as the correct parameter when the tag is written to the cell.
FormState	The form state key name for which you want to display the value in the current cell. For example, VPName. This is equivalent to using the GetFormState function to return the value of the specified form state key. If the specified form state key does not have a value, then the cell will be blank when rendered. The form must contain another component or tag that is configured to set the value of the form state key, or a GetFormState function that sets a default value for the form state key. The FormState parameter should only be used if the form is intended to be used as a dialog in the Excel Client or the Windows Client. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

Parameter	Description
SharedVariable	The shared variable name for which you want to display the value in the current cell. For example, PlanCode. This is equivalent to using the GetSharedVariable function to return the value of the specified shared variable. If the specified shared variable does not have a value, then the cell will be blank when rendered. The form must contain another component or tag that is configured to set the value of the shared variable, or a GetSharedVariable function that sets an initial value for the variable. The SharedVariable parameter should only be used if the form is intended to be used in an embedded form context (as either the parent form or a child form. For more information, see Sharing variables between parent and child forms.

The remaining parameters are common to all content tags. For more information on using these parameters, see Common parameters for content tags.

Parameters for spreadsheet-formatted grids

If you are using a legacy spreadsheet-formatted grid, the following parameters do not apply: Columns and ColumnStyle.

Spreadsheet-formatted grids support the following additional parameters: `Foreground=Color; Background=Color`.

For more information on using these parameters, see the topic on using the Format tag in Axiom Software Help.

Examples

```
[Format; Columns=2; ColumnStyle=col; TargetCell=N]
```

This example formats the current cell to span across two columns and to use the column style of **col**. The content for the cell is read from column N on the same row as this tag. This example is for a thematic grid.

```
[Format; Foreground=#FFFFFF;Background=#000000; TargetCell=Report!Z10]
```

This example formats the current cell to use the specified foreground and background colors instead of the cell formatting in the spreadsheet. The content for the cell is read from cell Z10 on the Report sheet.

Referencing cells in content tag parameters

When creating a content tag in a Grid data source, many tag parameters can accept special syntax to reference another cell in the workbook. This means that the tag will read the parameter value from the designated cell.

Using the special cell reference syntax for a parameter value can provide the following benefits:

- Cell references can eliminate the need to wrap the tag in a formula, when you want a parameter value to be able to change dynamically. Once a tag is wrapped in a formula, then it can no longer be edited using the Tag Editor or Data Source Assistant. Using cell references within the tag can make the tag easier to create and edit, versus using a formula.
- Cell references in the tag can optionally be relative to the current row, so that the tag references a designated target column on the current row. This can be useful when building out tags using an Axiom query.
- Cell references in the tag can optionally designate the target column relative to the column containing the tag. This can be useful in situations where columns may be inserted into a sheet, and these inserted columns may "break" references to specific columns.

Parameters that accept cell references

The following tag parameters accept cell references:

- Background
- Command
- ConfirmationMessage
- ColumnStyle
- DisplayCell
- Foreground
- HREF
- ImagePathURL
- MaxDate
- MinDate
- Placeholder
- TargetCell
- Text
- Symbol, Symbol-Checked, Symbol-Unchecked

For the TargetCell parameter, the content tag both reads and writes the value to the designated cell. For all other parameters, the content tag reads the value from the designated cell.

Cell reference syntax

For tag parameters that support a cell reference, the following syntax can be used:

Reference Type	Example	Description
Full cell reference	[Report!K10] or [K10]	The parameter value is read from the specified cell, either on another sheet or on the same sheet.

Reference Type	Example	Description
Column reference (current row)	[K]	The parameter value is read from the specified column, within the current row. For example, if the tag is on row 20, then the cell reference is resolved as K20.
Relative column reference (current row)	[+1] or [-1]	The parameter value is read from the specified relative column, within the current row. The relative column is specified as +N or -N columns from the current column, where N is a whole number. For example, if the cell reference is [+1], then the parameter value is read from 1 column to the right of the current cell. If the tag is in L20, then the cell reference is resolved as K20.

The cell references must be placed in brackets to signal to Axiom Software that the value is a cell reference and not the actual parameter value. For example, if you enter ColumnStyle=K, then Axiom Software would try to apply a column style named "K". But if you enter ColumnStyle=[K], then the column style is read from column K on the current row.

NOTE: For the parameters TargetCell and DisplayCell, it is not necessary to place the cell reference in brackets, regardless of which syntax you use. These parameters are specifically designed to use a cell reference, and therefore any entry will be evaluated as a cell reference, whether it has brackets or not.

Examples

```
[Symbol=fa-circle; ColumnStyle=[N]]
```

In this example, the column style is read from column N on the current row.

```
[Button; ButtonBehavior=Command; Text=[Variables!B3]; ButtonStyle=Push;]
```

In this example, the button text is being read from cell B3 on the Variables sheet.

```
[TextArea; TargetCell=+3; Placeholder=Enter a description;]
```

In this example, the text for the text box is being written to and read from the target cell located 3 columns to the right of the current cell. If the current cell is H10, the target cell is K10. The brackets can be omitted in this case for the TargetCell parameter, though they could also be included if desired.

Creating and editing content tags in Formatted Grids

You can create and edit content tags in Grid data sources manually, or you can use one of the following helper tools:

- The **Data Source Assistant** task pane
- The **Tag Editor** dialog

Using the Data Source Assistant

You can insert and edit any content tag using the Data Source Assistant task pane.

To insert a new tag:

1. Place your cursor in an empty cell where you want the tag to be placed.

   The cell must be within a tagged `[Row]` and `[Column]` within the data source (including `[Fixed]` rows and columns).

   > **TIP:** If you want to insert a tag into a cell that is not currently marked as a content row or column, then you can use the right-click menu to insert the tag. See the following section on using the Tag Editor dialog.

2. In the **Selection Editor** section of the task pane, select the desired **Tag Type**.

Once you select a tag type, the Selection Editor populates with the options for that tag type. You can then complete these options as desired. The tag will be built out as you make selections.

To edit an existing tag:

1. Place your cursor in the cell that contains the tag you want to edit.
2. In the **Selection Editor** section of the task pane, edit the tag options as desired.

As you make changes in the task pane, the tag in the cell is updated in real time. If you want to be able to preview your changes with the option to cancel, then do not use the Data Source Assistant task pane and instead double-click the cell to open the Tag Editor dialog.

NOTES:

- If the tag is created by a formula, then you cannot use the Data Source Assistant to edit the tag. The tag will not be recognized in the Selection Editor when you place your cursor in the cell. You must edit the tag directly in this case.
- By default, the parameters shown apply to thematic grids. However, if there is at least one spreadsheet-formatted grid within the file, then the Data Source Assistant shows the parameters that are applicable to spreadsheet-formatted grids.

Using the Tag Editor dialog

You can insert and edit any content tag using the Tag Editor dialog.

To insert a new tag:

1. Right-click the cell where you want to place the tag, then select **Insert Formatted Grid Tag >** ***TagName***.

   The cell can be anywhere within a form-enabled file; it does not have to currently be within a Grid data source. If the tag is not within a Grid data source when the form is rendered, it will be ignored.

2. In the **Tag Editor** dialog, complete the options for your selected tag and then click **OK**.

The tag is inserted into the current cell.

To edit an existing tag:

1. Double-click the cell that contains the tag you want to edit.
2. In the Tag Editor dialog, edit the tag options as desired, then click **OK**. If you click **Cancel**, your existing tag will be retained as is.

NOTES:

- If the tag is created by a formula, then you cannot use the Tag Editor to edit the tag. The tag will not be recognized in the Selection Editor when you place your cursor in the cell. You must edit the tag directly in this case.
- By default, the parameters shown apply to thematic grids. However, if there is at least one spreadsheet-formatted grid within the file, then the Tag Editor shows the parameters that are applicable to spreadsheet-formatted grids.

Common parameters for content tags

The following parameters are common to all content tags for thematic Formatted Grids. Any exceptions are noted.

Parameter	Description
ReadOnly	Optional. Specifies whether the control is "active" (True/False). The default value, False, means that the control is active and that the user can interact with it. If True, then the control becomes "frozen" and no further edits can be made. The control will display the current value of the target cell. This parameter can be used to control whether a user can edit the cell. Generally speaking, this parameter would only be used within a formula to dynamically enable / disable the check box. This parameter only applies to tags that generate interactive controls: CheckBox, DatePicker, Select, TextBox.

Parameter	Description
AutoSubmit	Optional. Specifies how interactive controls submit values back to the source file: • `Enabled`: The control uses auto-submit behavior, regardless of whether the grid is set to auto-submit. Changed values are submitted as soon as they are changed. • `Disabled`: The control does not use auto-submit behavior, regardless of whether the grid is set to auto-submit. Changed values are not submitted until another interactive control triggers a submit. • `Grid`: The control honors the auto-submit behavior as configured for the grid. If omitted, the default behavior is Grid. This parameter only applies to tags that generate interactive controls: CheckBox, DatePicker, Select, TextBox.
Tooltip	Optional. Specifies tooltip text to display when a user hovers the cursor over the cell contents. Alternatively, you can use a bracketed cell reference to read the tooltip text from the referenced cell. This approach is useful if you want to dynamically determine the text, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters. This parameter does not apply to Sparkline tags.
Columns	Optional. Specifies how many columns the cell contents will span in the grid. If this parameter is omitted or set to 1, then content generated by the tag will only span the current column. If you want the content to span multiple columns, enter a number such as 2 to span 2 columns. The column span extends to the right. **NOTE:** The row and column styles used in the grid impact how the column span displays. For example, if the content in the starting column is left-aligned and does not naturally exceed the width of the starting column, then the spanned columns will simply be blank because no content is extending to those columns. However, if the content is long enough to extend out of the starting column, or if the content has external borders (such as a text box), or if the content is center-aligned or right-aligned, then content will display in the spanned columns.

Parameter	Description
ColumnStyle	Optional. Specifies one or more column styles to apply to the current cell. The specified styles override the current column styles set by the `[ColumnStyle]` tag, but only for the current cell contents (including the column span, if defined). The next cell down will not inherit the styles specified for this cell; the next cell will revert to using the currently applied column styles. Enter one or more valid column style names, separated by commas. If you are using the Data Source Assistant / Tag Editor, you can click the **[...]** button to open the **Choose Styles** dialog and select from available styles.The available styles depend on the skin specified for the form. For more information, see Using row and column styles in a thematic grid. Alternatively, you can use a bracketed cell reference to read the style from the referenced cell. This approach is useful if you want to dynamically determine the style, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.

Note the following exceptions:

- For the TextArea tag only, the Columns parameter applies to both spreadsheet-formatted grids and thematic grids. For all other tags, it only applies to thematic grids. In spreadsheet-formatted grids, this parameter can be used as an alternative to merged cells, which are not supported in Formatted Grid components.
- The AddRows tag is not supported for use in thematic grids and therefore does not use the thematic-only parameters.

Setting up drilling for Formatted Grids

You can enable drilling in Axiom forms by setting up drill-down drilling for Formatted Grid components. This feature is similar to the drill-down feature that is available for spreadsheet Axiom files in the Desktop Client. Users can "drill down" a row in the grid to see the data in that row at a different level of detail.

For example, the Formatted Grid component can be used to display an Income Statement for the consolidated organization. Users can drill rows in this report, such as a revenue row, to see the data broken out by region, entity, VP, or some other grouping.

When drilling a Formatted Grid component in an Axiom form, the drill results are presented in a new tab. This tab contains a single, simple grid that displays the drilling data. The rows of the drilling data are determined by the selected drilling level. For the data columns, you specify which columns from the original grid are shown in the drill results.

This feature is the only standard way to set up drilling for Axiom forms, however, there are many other ways that you can create custom drilling options within an Axiom form. For example, you can pass a value (such as a selected region or entity) to a second Axiom form via hyperlink, and then display data for that passed value within the second form. Or you can use a selection within the current form to change the data shown in a grid or chart, or to show different grids and charts that are filtered by the selected value.

Requirements to enable data drilling in Formatted Grids

Axiom forms support drill-down drilling by using Formatted Grid components. This drilling feature is not automatically available; it must be specifically enabled and set up within the form.

Drilling requirements

In order to enable drill-down drilling for a Formatted Grid component, the grid must meet the following requirements:

- The grid must be a thematic grid (**Grid Formatting** set to **Thematic**). Spreadsheet-formatted grids do not support drilling.
- The Grid data source must contain drilling tags to enable the drill, and to flag certain rows and columns as part of the drill configuration. Rows can be enabled or not for drilling, and data columns can be included in the drill results or excluded. For more information on how to set up these tags, see Configuring the Grid data source for data drilling.
- The data presented in the Grid data source must be from an Axiom query. GetData functions cannot be drilled in Axiom forms.
- The default drilling behavior assumes that the grid does *not* use the `[RowID]` tag. If you want to use the `[RowID]` tag with the grid, then you must also set up a Button component with the Drill button behavior. For more information, see Using a Button component to drill a Formatted Grid.

The Axiom query used to populate the grid must be set up as follows:

- The Axiom query must be active and enabled for drilling. This means that the **Active** and **Drillable** properties must be set to **On** for the Axiom query in the default Control Sheet. Additionally, only vertical queries are eligible for drilling.
- The Axiom query must have a defined in-sheet calc method, even if the query is update-only. This is because the calc method is carried over to the drill results and used to populate the result grid.

- The data in the Axiom query must have associated hierarchies so that users can select the drilling level from a hierarchy, or you must create a DrillLevels data source within the file to specify the available drilling levels. For more information on how to specify the available drilling levels for the drill (hierarchies or data source), see Configuring the Grid data source for data drilling.

  If there are no associated hierarchies and no DrillLevels data source, then the data cannot be drilled because there are no drilling levels to choose from.

  NOTE: The general drilling options of **All Detail** and **Choose Columns** are *not* available when drilling in Axiom forms. These options are only available when performing a drill down in a spreadsheet Axiom file.

Additionally, drilling is not available in all browsers and/or may require certain browser configuration changes:

- Some browsers may require pop-ups to be allowed for the Axiom Software site in order to perform drilling in an Axiom form.
- Drilling does not work on the iPad, using either the Safari browser or the Axiom Software iPad app.

Axiom query design considerations

When configuring the Axiom query, note the following design considerations that may impact the drill results.

Configuration	Drilling behavior
Multiple-row calc methods	If the Axiom query uses a multiple-row calc method, the drill results present the drilled data for all rows of the calc method, not just the row that was drilled. Additionally, the number formats for all rows of the drill results will be taken from the first row of the calc method. For example, this means that if the first and second rows are formatted as currency and the third row is formatted as percent, the third row will be formatted as currency in the drill results.
Calc method library	If the query uses a calc method library instead of an in-sheet calc method, then the drill results use the calc method that is applied to the row being drilled (based on the calc method validation column). If the sheet does not use validation (and therefore there is no calc method validation column), then the **Default Calc Method** for the query is used. If the sheet does not use validation and there is no default calc method, then no calc method is applied to the drill results. This means that the drill results will be empty, so it is required to either use validation or define a default calc method. This only applies to file group files that have access to a calc method library, such as plan files and file group utilities.

Configuration	Drilling behavior
Nested Axiom queries	You can drill nested Axiom queries. A nested Axiom query is where the in-sheet calc method of one query is used to build out a second "child" query. The drill results will return data for the Axiom query that the drilled row belongs to. For example, if AQ1 builds out multiple data ranges for AQ2, and you drill a row within an AQ2 data range, then the drill results will be for AQ2. Results depend on how the queries are set up, but in most cases this should return the drill results that you are expecting.
Parallel Axiom queries	You can drill parallel Axiom queries. A parallel Axiom query is where multiple Axiom queries update the same set of rows. The AQ# tags for each query are on the same row. In this case, the drill results will be only for whichever query is listed first on the Control Sheet. For example, if AQ1 and AQ2 update the same set of rows, then the drill results will be for AQ1. Any data for AQ2 will be ignored. Results depend on how the queries are set up; some configurations may provide useful drill results, while others will not. **NOTE:** If the first query context is invalid for drilling (for example, if AQ1 is disabled or **Drillable** has been set to **Off**), then Axiom Software will attempt to drill on the next relevant query—in this example, AQ2.
Update-only Axiom queries	If the refresh behavior for an Axiom query is set to update-only, that query must still have a defined calc method in order for the query data to be drillable. Even though the original Axiom query does not use the calc method, the calc method will be used in the modified query that provides the drill results. The calc method must contain a `[Row]` tag in the Grid control column, so that when the drill result query is run, the query rows will show in the result grid. The calc method must also contain any formulas that you want to display in the drill results.
System tables	Axiom queries to system tables (such as Axiom.Aliases) cannot be drilled.
Alternate aggregations	Use of AxAggregate on a column does not prevent drilling, but in some cases it does not return the values that you might expect on the drill sheet. For example, if AxAggregate(Avg) is used, the average values on the drill sheet will not correspond with the average value on the original row. This is because the sum by on the drill sheet is at a different level, resulting in different average calculations per record.
Data Conversions	If data conversion is enabled for the Axiom query, the conversion also applies to the drill down data.
Segmented data	Segmented data is not supported for use with drill-down drilling.

Configuring the Grid data source for data drilling

In order to enable a Formatted Grid component for drill-down drilling, the Grid data source must contain the required drilling tags. These tags are used to:

- Specify the available drilling levels for users to choose from. You can use hierarchies to provide drilling levels, or you can create a DrillLevels data source that specifies grouping columns to use for drilling.
- Specify the rows that are eligible for drilling. Users can click any flagged row to view that row's data at a different level of detail.
- Specify the columns to be included in the drill results. When the drill result page is created, the flagged columns will be included in the drilling data.

The drilling tags must be added to the Grid data source. Typically these tags are placed before any data rows and columns, along with the other "setup" tags (such as `[RowStyle]` and `[ColumnWidth]`), but they can be present anywhere within the data source.

Drilling tag summary for Grid data sources

Tag Type	Tag Syntax

Row tags
(placed in Grid control column)
	`[DrillDownColumns; HierarchyOrDataSourceName]`
	This tag enables drilling for the grid and also specifies the source of the drilling levels (hierarchy or DrillLevels data source). See Using hierarchies for drilling levels or Using a DrillLevels data source for drilling levels for more information.
	To flag columns to include in the drill results, place the tag `[DrillColumn]` in this row.
	`[DrillHeaders]`
	Optional. Defines header text for each drill column to display in the drill results. If omitted, the text from `[Fixed]` rows will be used.

Column tag (placed in Grid control row)	`[DrillDownRows]` To flag rows that are eligible for drilling, place the tag `[DrillRow]` in this column.

See Drilling example for a screenshot of these tags placed within a Grid data source.

Flagging columns to include in the drill results

The `[DrillDownColumns]` row is used to flag data columns that you want to be included in the drill results. When a user drills, the drilling data consists of the user's selected drilling level (such as regions or departments), plus all columns that are flagged to be included from the original grid. If you do not flag any columns, then the drill results will only show columns for the selected drilling level, with no corresponding data.

To include a column in the drill results, enter the tag `[DrillColumn]` into the `[DrillDownColumns]` row. Any type of column can be included, whether it contains queried data or formulas. Columns will be displayed in the same order as they are shown in the original grid.

Generally speaking, the following columns should not be flagged as drill columns:

- Columns that show the current Axiom query "sum by" level, and any corresponding description columns. When drilling, the data is shown at a different level, so these columns will not be relevant in the drill results. The dimensionality of the original row is noted on the drill results page for reference.
- Columns that contain formulas that reference the columns showing the current Axiom query "sum by" level. These formulas will not be re-evaluated against the drilling level; instead they will continue to show the current results, which will likely not be relevant in the drill results.
- Columns that contain content tags. The drill results grid does not support these tags, so they will display as the raw tag instead of as the desired result. The exception to this rule is the Format tag, which will be converted to its target text (however, all other formatting in the tag will be ignored).
- Columns that contain editable cells (unlocked cells), unless you want to display the current contents in the drill results grid as read-only. Cells are not editable in the drill results grid.

For more information about how columns are displayed in the drill results, see Presentation of drill results when drilling Formatted Grids.

Flagging rows to be eligible for drilling

The `[DrillDownRows]` column is used to flag data rows that are eligible to be drilled. The drill-down feature is based on taking a specific row in the grid and showing the data in that row at a different level of detail. For example, a user may want to drill a revenue row in an Income Statement so that they can see how the revenue numbers break out by groupings such as region, entity, or VP.

To enable a row for drilling, enter the tag `[DrillRow]` into the `[DrillDownRows]` column. You can drill any row that is populated by the Axiom query, whether the data is populated via rebuild or update behavior. If the Axiom query uses rebuild behavior, then you must make sure that the in-sheet calc method for the query includes the `[DrillRow]` tag (just as it must also include the `[Row]` tag so that it displays in the grid).

All other rows in the grid—such as header rows, spacer rows, or subtotal/total rows—should not be flagged as drill rows because those rows are not part of the Axiom query and therefore the drill will fail.

Using hierarchies for drilling levels

You can configure the drill so that users select a drilling level from one or more hierarchies that are associated with the Axiom query data. This is similar to how drill-down drilling works for spreadsheet Axiom files in the Desktop Client. However, when using hierarchies to drill in an Axiom form, you can specify which hierarchies you want to be available to the user.

If you want to use hierarchies to define the drilling level, then you can complete the *HierarchiesOrDataSource* parameter on the `[DrillDownColumns]` tag as follows:

Hierarchy Option	Description	Example
<Blank>	If the *HierarchiesOrDataSource* parameter is omitted from the tag, then all relevant hierarchies are shown to the user (based on the primary table of the Axiom query being drilled).	`[DrillDownColumns]` Displays all relevant hierarchies. These are the same hierarchies that show when performing a drill-down on a spreadsheet Axiom file in the Desktop Client.

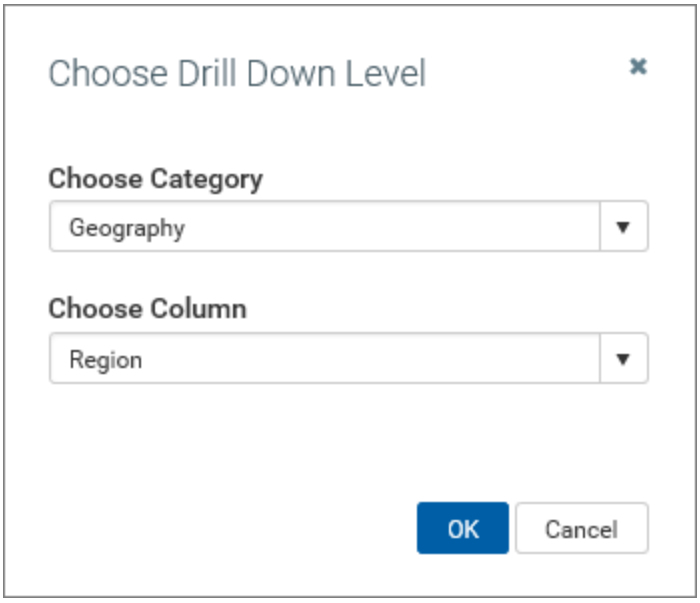
Table	Enter a table name to display all hierarchies defined for that table.
	You can also enter multiple table names, separated by commas. The dialog will display all hierarchies defined for all listed tables.
		`[DrillDownColumns; Dept]`
		Displays all hierarchies defined on the Dept table.
		`[DrillDownColumns; Dept,Acct]`
		Displays all hierarchies defined on the Dept table and the Acct table.

Table:*Hierarchy*	Enter a table name plus a hierarchy name to only show the specified hierarchy.
	You can also enter multiple table:hierarchy pairs, separated by commas. The dialog will display all specified hierarchies.
	NOTE: Keep in mind that if the query data contains multiple paths to the hierarchy columns, the same hierarchy will show multiple times (once for each valid path). You can specify a Table.Column name instead if you want the drill to always go through a specific path (see next row).
		`[DrillDownColumns; Dept:Geography]`
		Displays the Geography hierarchy defined on the Dept table.
		`[DrillDownColumns; Dept:Geography, Acct:Type]`
		Displays the Geography hierarchy defined on the Dept table and the Type hierarchy defined on the Acct table.

Hierarchy Option	Description	Example
Table.Column: *Hierarchy*	Enter a Table.Column name plus a hierarchy name to only show the specified hierarchy path, and to apply the selected hierarchy level in the context of the table.column. This may be helpful when the query data contains multiple paths to the hierarchy columns, which by default causes hierarchies to show multiple times.	`[DrillDownColumns; Dept.Region:Region]` Displays the Region hierarchy on the Region table, where `Dept.Region` looks up to the Region table. Additionally, in this example the resulting drilling level will be defined as `Dept.Region.RegionType` instead of just `Region.RegionType` (where RegionType is a level in the Region hierarchy).

If you configure the drill to use specific hierarchies, you must make sure that hierarchy is valid within the context of the Axiom query. The hierarchy must be on a lookup reference table for the primary table of the query. Additionally, if the query uses multiple data tables, then the hierarchy must be on a shared lookup reference table for all of the data tables in the query.

When hierarchies are used, users first select a category (the hierarchy) and then select a column in the hierarchy. In the following example, the user has selected the Geography hierarchy and then the Region column, so the drilling data will use regions as the rows.



If only one hierarchy is available, then the user does not have to select the hierarchy. Instead, the columns in the hierarchy are presented in the same way as the options from a DrillLevels data source (as shown in the next section).

For more information on how the drilling data is created based on the selected drill level, see Presentation of drill results when drilling Formatted Grids.

For more information on creating hierarchies, see the *System Administration Guide*.

Using a DrillLevels data source for drilling levels

You can configure the drill so that users select from a list of defined drilling choices, where each drilling choice corresponds to a column that you want to allow users to drill by. This provides you with full control over how the drilling levels are presented and which columns can be used to drill.

To do this, create a DrillLevels data source on any sheet within the form-enabled file. The DrillLevels data source defines the columns that can be used to drill, and the display text to show to users for each option.

Then, specify the data source name within the *HierarchiesOrDataSource* parameter on the `[DrillDownColumns]` tag. For example:

```
[DrillDownColumns;DeptDrill]
```

Where DeptDrill is the name of the DrillLevels data source.

The tags for the DrillLevels data source are as follows:

Primary tag

`[DrillLevels;`*`DataSourceName`*`]`

The DataSourceName uniquely identifies this data source so that it can be placed in the `[DrillDownColumns]` tag. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

NOTE: You should avoid giving the data source the same name as a table in your system with hierarchies. If the name placed in the `[DrillDownColumns]` tag can be resolved as either a data source or a table, Axiom Software will use the data source, which may cause confusion. For example, instead of naming the data source Dept, name it DeptDrill.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[DrillItem]

Each row flagged with this tag specifies a drilling option to present in the Drill Level selection dialog.

Column tags

[Label]

Defines the display name of each item in the Drill Level selection dialog.

[Column]

Defines the Table.Column to use for the drilling level when the corresponding label is selected. For example, if the column is Dept.Region, then the drilling data is by region. Multiple-level lookups can be used.

It is up to the form designer to ensure that each column listed is valid and relevant to the data that can be drilled. Generally speaking, if the data query only uses one data table, then any column in the table itself as well as any column in lookup reference tables can be used. If the data query uses multiple data tables, then only shared lookup reference tables can be used. Other columns may return unexpected drilling data, or may result in drilling errors.

NOTES:

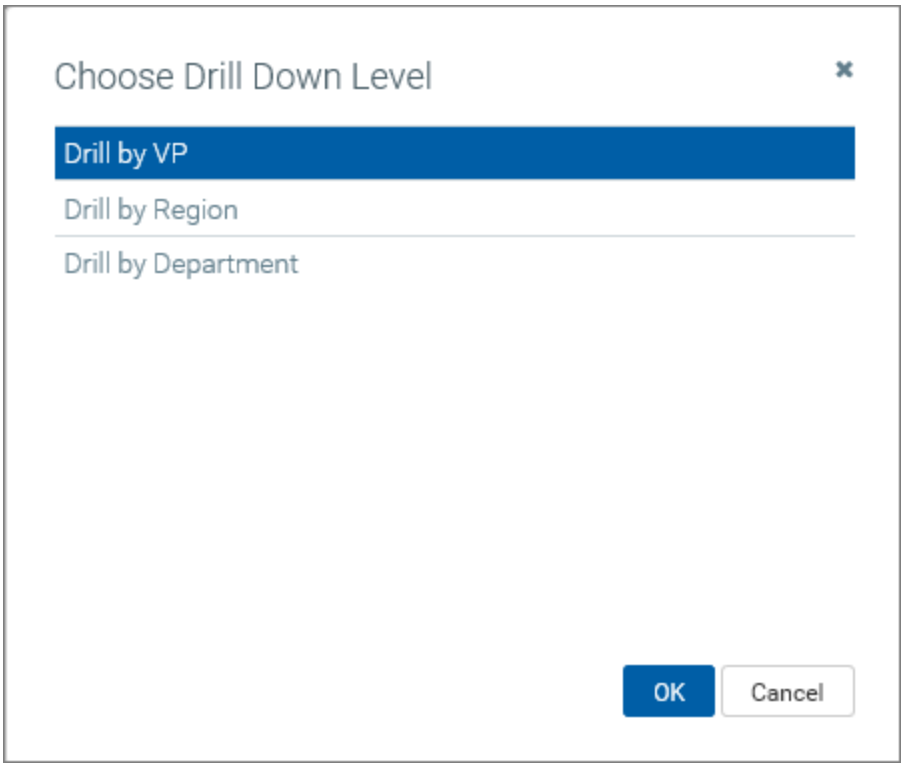
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Drill Levels**. You can also highlight a range of data first and then use the wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The following example shows a DrillLevels data source:

	A	B	C	D	E
6					
7			[DrillLevels;DeptDrill]	[label]	[column]
8			[DrillItem]	Drill by VP	Dept.VP
9			[DrillItem]	Drill by Region	Dept.Region
10			[DrillItem]	Drill by Department	Dept.Dept

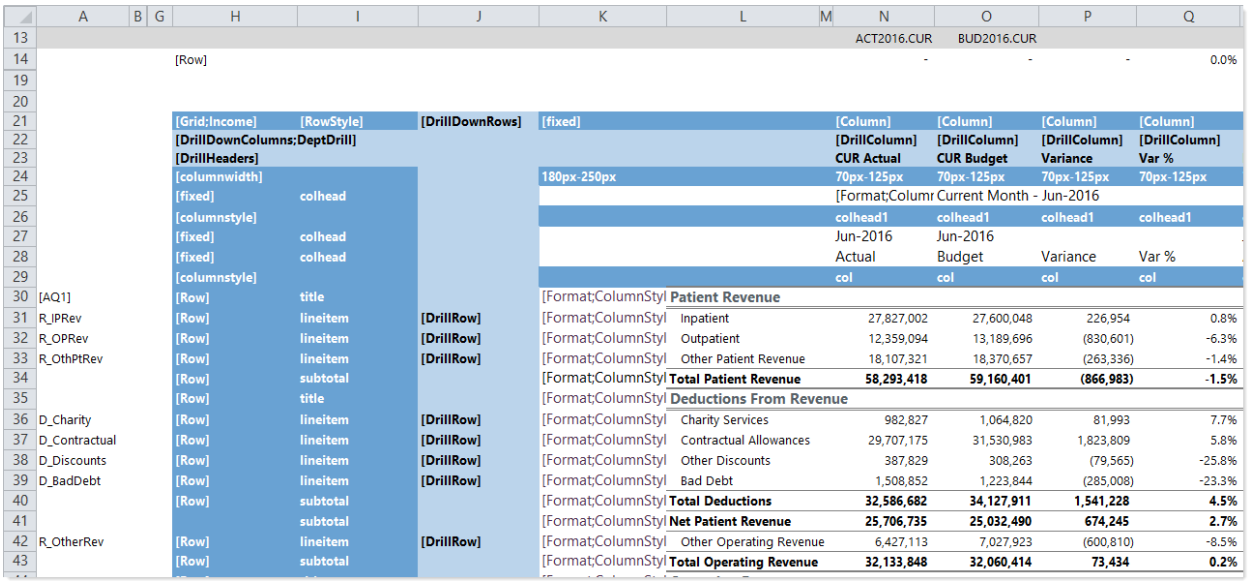
When a user initiates a drill, the drilling items are displayed in the **Choose Drill Down Level** dialog as shown in the following screenshot. Only the label displays; the column is not shown (unless you include the column name in the label).



The drill results are then created using the corresponding column for the selected label. For more information on how the drilling data is created based on the selected drill level, see Presentation of drill results when drilling Formatted Grids.

Drilling example

The following example shows a Grid data source that has been tagged to allow drilling. The columns with drilling tags have been shaded in light blue to contrast them with the regular Grid tags.

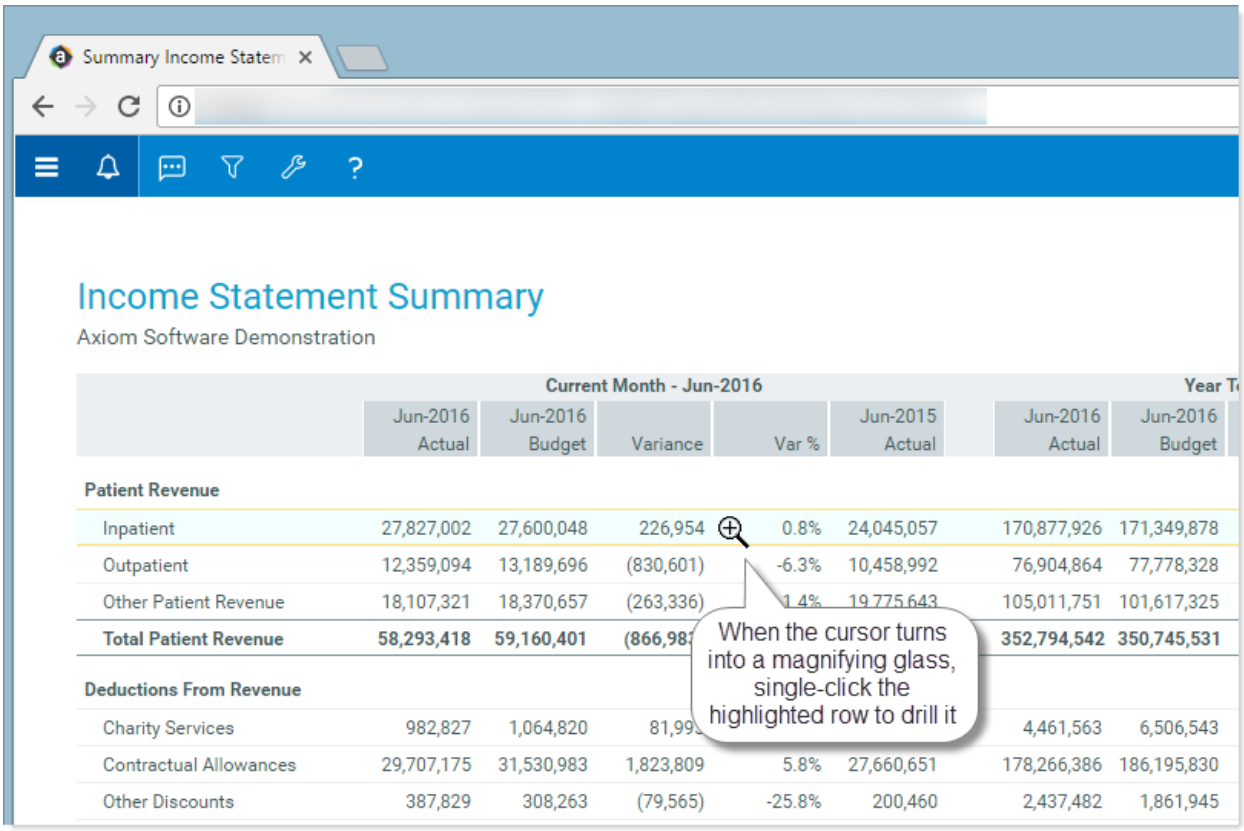


	A	B	G	H	I	J	K	L	M	N	O	P	Q
13										ACT2016.CUR	BUD2016.CUR		
14				[Row]						-	-	-	0.0%
19													
20													
21				[Grid;Income]	[RowStyle]	[DrillDownRows]	[fixed]			[Column]	[Column]	[Column]	[Column]
22				[DrillDownColumns;DeptDrill]						[DrillColumn]	[DrillColumn]	[DrillColumn]	[DrillColumn]
23				[DrillHeaders]						CUR Actual	CUR Budget	Variance	Var %
24				[columnwidth]			180px-250px			70px-125px	70px-125px	70px-125px	70px-125px
25				[fixed]	colhead					[Format;Columr	Current Month - Jun-2016		
26				[columnstyle]						colhead1	colhead1	colhead1	colhead1
27				[fixed]	colhead					Jun-2016	Jun-2016		
28				[fixed]	colhead					Actual	Budget	Variance	Var %
29				[columnstyle]						col	col	col	col
30	[AQ1]			[Row]	title		[Format;ColumnStyl	Patient Revenue					
31	R_IPRev			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Inpatient		27,827,002	27,600,048	226,954	0.8%
32	R_OPRev			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Outpatient		12,359,094	13,189,696	(830,601)	-6.3%
33	R_OthPtRev			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Other Patient Revenue		18,107,321	18,370,657	(263,336)	-1.4%
34				[Row]	subtotal		[Format;ColumnStyl	**Total Patient Revenue**		**58,293,418**	**59,160,401**	**(866,983)**	**-1.5%**
35				[Row]	title		[Format;ColumnStyl	Deductions From Revenue					
36	D_Charity			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Charity Services		982,827	1,064,820	81,993	7.7%
37	D_Contractual			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Contractual Allowances		29,707,175	31,530,983	1,823,809	5.8%
38	D_Discounts			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Other Discounts		387,829	308,263	(79,565)	-25.8%
39	D_BadDebt			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Bad Debt		1,508,852	1,223,844	(285,008)	-23.3%
40				[Row]	subtotal		[Format;ColumnStyl	**Total Deductions**		**32,586,682**	**34,127,911**	**1,541,228**	**4.5%**
41					subtotal		[Format;ColumnStyl	**Net Patient Revenue**		**25,706,735**	**25,032,490**	**674,245**	**2.7%**
42	R_OtherRev			[Row]	lineitem	[DrillRow]	[Format;ColumnStyl	Other Operating Revenue		6,427,113	7,027,923	(600,810)	-8.5%
43				[Row]	subtotal		[Format;ColumnStyl	**Total Operating Revenue**		**32,133,848**	**32,060,414**	**73,434**	**0.2%**

- Row 22 contains the `[DrillDownColumns]` tag. In this row, you can see that columns N-Q are flagged to be included in the drill results. This tag also defines the source of drilling levels as a DrillLevels data source named DeptDrill (defined on a different sheet).
- Column J contains the `[DrillDownRows]` tag. Because this Axiom query is an update-only query, it is set up so that only the rows updated by the query are flagged as drillable rows. The various subtotal and title rows are not included. If the query was a rebuild query, then you would place the `[DrillRow]` tag in the calc method so that all data rows are flagged as drillable when the query is populated.
- Row 23 contains a `[DrillHeaders]` tag to define headers for the drill results. This is optional; alternatively we could have used the headers that are already in the `[Fixed]` rows.

Also notice the `[Row]` tag in row 14, which is the in-sheet calc method for the Axiom query. This is standard procedure for rebuild Axiom queries, so that the data rows are displayed in the grid when the query is refreshed. However, when setting up an Axiom query for drilling, you must also do this for update-only Axiom queries. Even though the calc method is not used when refreshing the update-only query, it will be used to populate the drilling results. If the calc method does not contain a `[Row]` tag, the drill result grid will be blank because it has no rows to display.

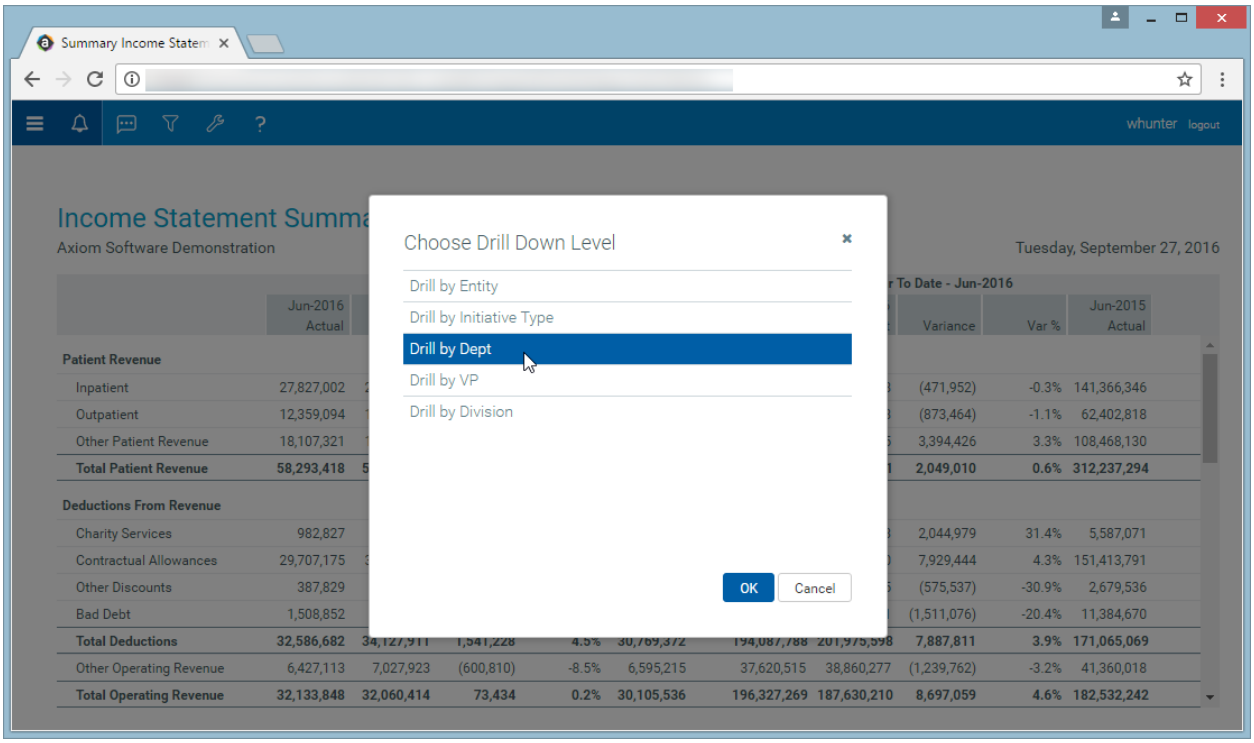
When this file is viewed as a form, users can single-click any drillable row to view that row's data at a different level of detail. When a user hovers over a drillable row, the row becomes highlighted and the cursor turns into a magnifying glass to indicate that drilling is available.



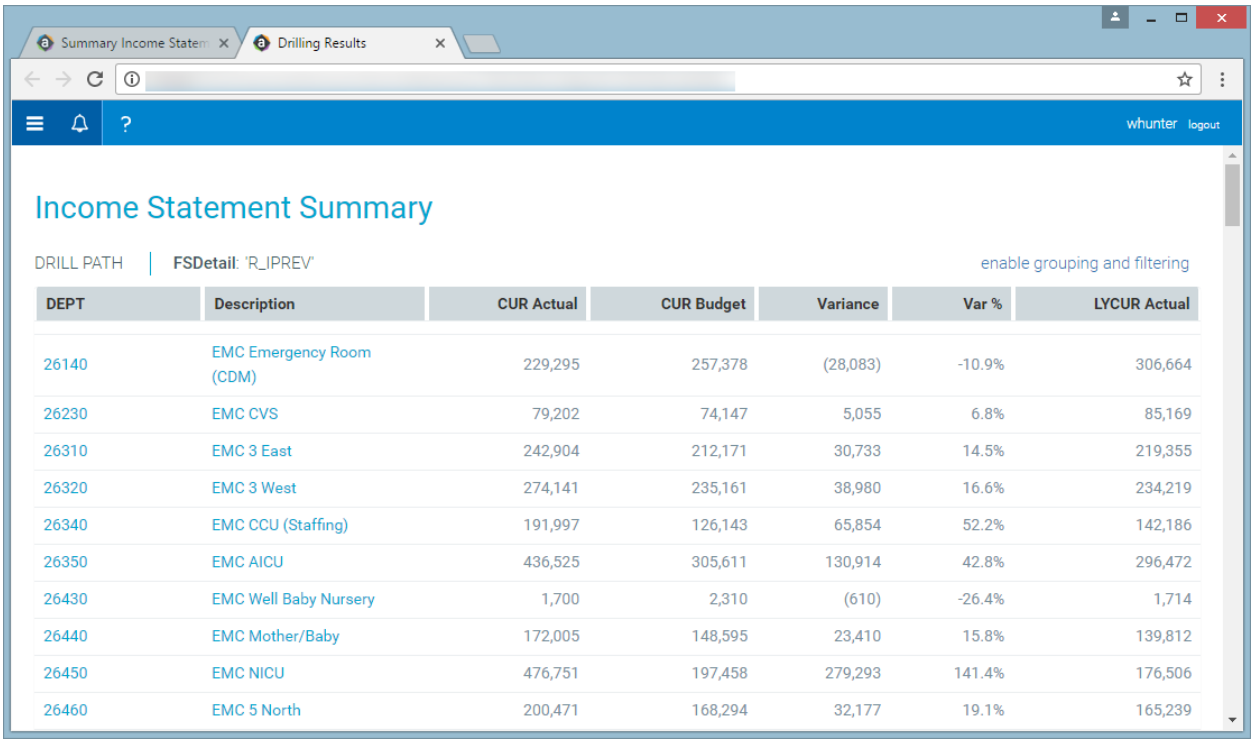
User hovers over drilling-eligible row

NOTE: The single-click drill behavior assumes that the grid is not using the `[RowID]` tag to enable selecting rows. If you want to use the `[RowID]` tag and still allow users to drill the grid, then you must also set up a Button component that uses the Drill button behavior. Users can select a row in the grid, and then click the button to drill that row. For more information, see Using a Button component to drill a Formatted Grid.

Once the user has initiated the drill, a dialog opens to display the available drilling levels. In this example, these are the drilling levels defined in the DrillLevels data source named DeptDrill.



After the user selects a drill level (Drill by Dept in this case), then a second tab opens to display the drill results. In this example, the data for the Inpatient Revenue row is now shown grouped by departments.

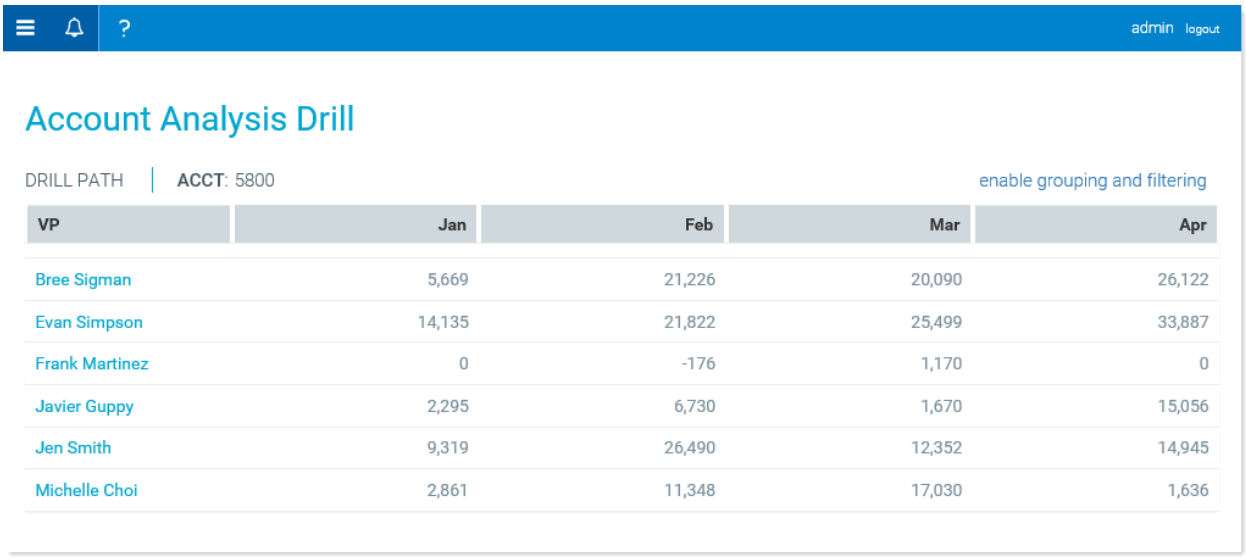


The data columns in the drill results are the columns that were flagged as drill columns in the original grid. In this example, only the data columns for the current month (the first section of columns) were flagged as drill columns, so that is the data shown in the drill results. The year-to-date columns were not flagged as drill columns, so they are omitted from the drill results. As this example illustrates, the form designer determines which drill levels are available to users, and what data will be shown in the drill results.

Presentation of drill results when drilling Formatted Grids

When a user drills a Formatted Grid component, the drill results are presented in a separate page (tab in the browser). This drill results page consists of the following:

- A page title and subtitle that identify the drilling data.
- An unformatted grid to display the drilling data.



Example drill result page

The contents of this page are created using a separate file (the "drill result" file). This file contains a copy of the sheet being drilled, renamed as FormsDrillResults in the drill result file. This sheet is automatically adjusted as necessary to arrive at the drill results. It can be helpful to understand how the drill results are created and how they display, in order to set up your Axiom query and Grid data source appropriately.

When the drill result file is created, if the sheet being drilled contains formulas to other sheets in the file, these formulas are converted to values. The other sheets in the file are not copied to the drill result file.

Page title and subtitle

The page title is determined as follows:

- If **Title Text** is defined for the source Formatted Grid component that was drilled, that text displays as the page title. It does not matter whether the title bar is shown on the component; if the text is defined, it will be used.
- Otherwise, the title text is **Drill Result for *FileName***.

The page subtitle shows the text **Drill Path** followed by the dimensionality of the row being drilled. For example, if you drill a row that contains data for the revenue account category, the subtitle is **Drill Path | Category: Revenue**.

The browser tab text is always **Drilling Results**.

Drilling data

The drilling data is obtained using a modified version of the original Axiom query that is being drilled. This query is created as Axiom Query #1 in the drill result file. The settings for the Axiom query being drilled are copied to Axiom Query #1, and then this query is adjusted as follows:

- The Axiom query is converted to rebuild refresh behavior. This is why the in-sheet calc method must still be defined if you are drilling an update-only Axiom query. The calc method must contain a `[Row]` tag in the appropriate column, so that when the query is refreshed, the rows are flagged to be included in the drill results grid. The calc method must also contain any formulas that you want displayed in the drill results.
- The "sum by" for the Axiom query is set to the selected drill level. For example, if the drill level is Dept.Dept, then the sum by is set to Dept.Dept. The sort level is also set to the selected drill level.
- The column for the selected drill level is added to the Axiom query field definition. For example, if the drill level is Dept.Dept, then that column is added to the field definition. Also, because Dept.Dept is the key column of a reference table, the designated description column for the table (for example, Dept.Description) is also added to the field definition. All new columns are added to the end of the current used range, so that they will not overwrite anything in the sheet.
- The dimensionality for the row being drilled is added to the Data Filter for the Axiom query. For example, if the row you are drilling contains data for the revenue account category, the following would be added to the filter: `ACCT.Category='Revenue'`. Any existing Data Filter for the Axiom query is also retained.

  NOTE: If a Sheet Filter was configured on the sheet being drilled, that Sheet Filter is also copied to the drill result file.

- In the sheet copied from the source file, all Axiom query data range tags are cleared, and new tags are placed for Axiom Query #1 in column A, in the row where the first tags were found. This is so that all previous Axiom query data is cleared from the sheet when the drill result query is run.

After the Axiom query is refreshed, this data is used to populate the drill results grid. The column or columns for the selected drill level are shown first, followed by all columns that were flagged with `[DrillColumn]` in the original grid. For example, if the selected drill level is Dept.Dept, then the Dept and Description columns display first, followed by the drill columns.

The drill results grid is a simple grid. None of the formatting defined in the original grid will carry over to the drill results grid.

Grid headers

The drill results grid contains one or more header rows for column headers. Header text for the drill level columns is auto-generated using their column names. Header text for all other columns is determined as follows:

- If a `[DrillHeader]` row exists, the text in that row is used for column headers.
- Otherwise, the text in the `[Fixed]` rows of the original grid is used for column headers.
- If no `[DrillHeader]` or `[Fixed]` rows exist in the Grid data source, then the column headers will be labeled with generic text such as "Column5".

Miscellaneous

- Generally speaking, content tags within the Grid data source are not recognized by the drill results grid. These tags will display as the raw tag within the drill results grid. The sole exception is the Format tag.
- Format tags are converted to display their target text in the drill results grid. However, all formatting defined in the format tag is ignored. This includes column spans. The target text will display in the first cell of the span, and all other cells that would normally be covered by the span will instead display their contents as normal (assuming that the columns are flagged as drill columns). This means that the target text should not be placed within the cells of the span; instead the target text should be placed outside of the grid drill columns.

Using a Button component to drill a Formatted Grid

You can optionally use a Button component to initiate a drill on a row in a formatted grid, instead of using the default behavior of single-clicking on a row. This approach is necessary if you want to use the `[RowID]` tag with the grid, so that users can select rows.

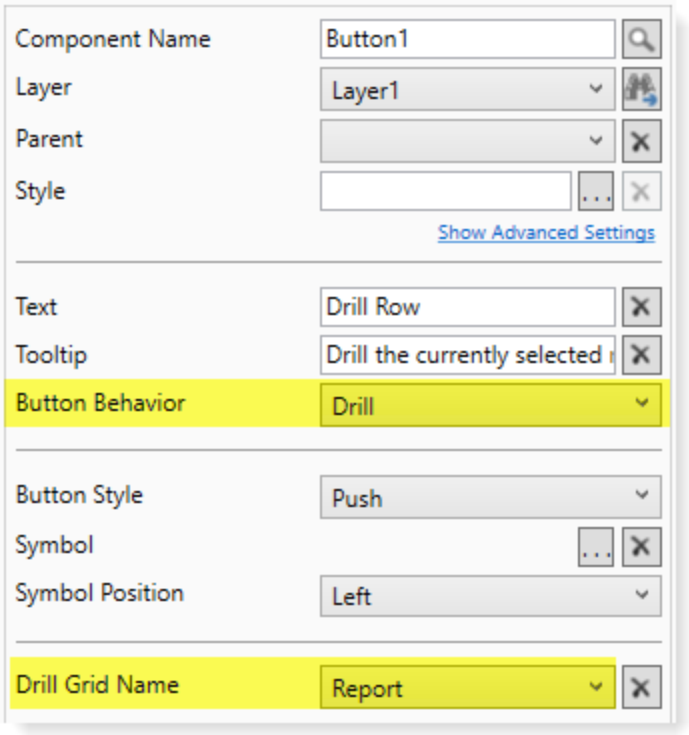
The requirements for using a Button component to drill a Formatted Grid are as follows:

- The grid must contain a `[RowID]` tag, and all drillable rows must be populated with unique IDs.
- **Auto-Submit** must be enabled for the grid. The drill button will not be enabled until a drillable row has been selected and that selection is submitted back to the source file.
- The form must contain a Button component that is configured to use the **Drill** button behavior. See the following section for more details on how to configure the drill button to drill the desired grid.
- All regular drilling requirements still apply when using a Button component to drill. The button simply changes the way that the user initiates the drill.

Configuring a Button component to use the Drill button behavior

To create a drill button, place a Button component on the form canvas and then configure the settings as follows:

- For the **Button Behavior**, select **Drill**.
- For the **Drill Grid Name**, select the name of the Formatted Grid component that you want to drill using the button. In the example below, the grid has been named "Report."

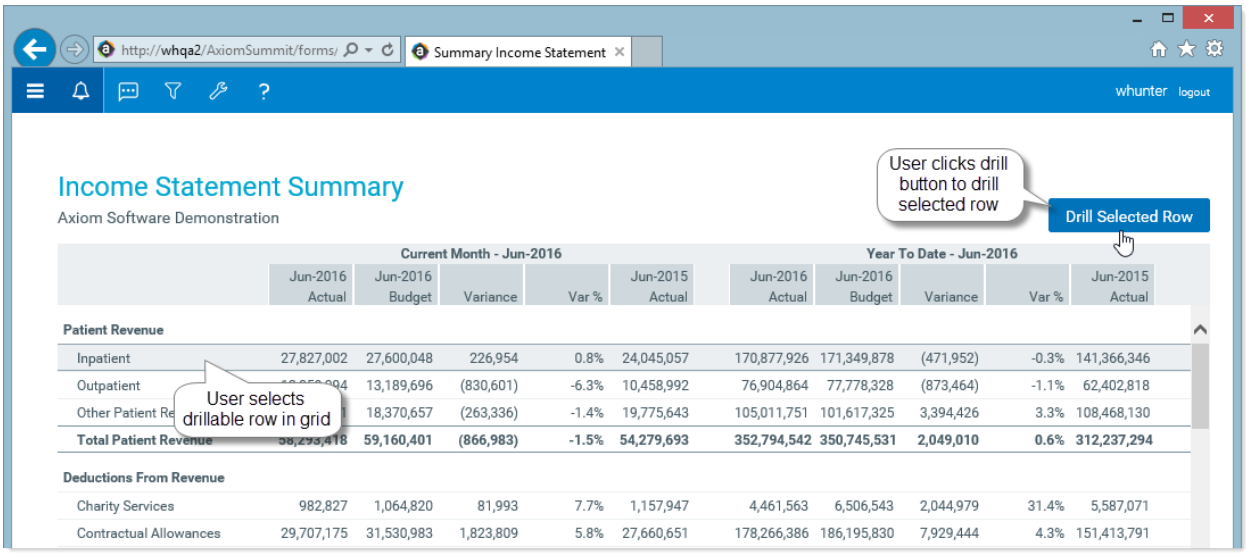


You can set the button style and the text as desired. If the button style is **Push** or **Link**, you probably want to define the button text to something like "Drill Row" or "Drill Current Row". If you are using an **Image** button, then the form may need separate explanatory text either within the grid itself or using a separate Label component.

Button components that use the Drill button behavior do *not* trigger an update of the current form. The only action performed by the button is to initiate the drill. Once the button is clicked, the behavior is exactly the same as when using the default single-click behavior for drilling.

Drilling example when using a drill button

When using a drill button, the user must first select a drillable row in the grid. The grid must be set to auto-submit so that the row ID for the selected row is written back to the Selected Row ID field for the grid.



When the drill button is clicked, Axiom Software reads the Selected Row ID for the grid, and initiates a drill based on that row. The drilling choices are displayed as normal, and the separate drill results page is opened as normal.

The drill button is disabled if any of the following are true:

- The specified grid for the drill does not currently have a Selected Row ID.
- The grid has a Selected Row ID, but that row is not flagged as a drillable row.

Editing grid contents in a spreadsheet editor

You can provide form users with a spreadsheet-style editor to edit the contents of a Formatted Grid component. This spreadsheet editor opens as a modal dialog in the browser, so users do not have to exit the current form in order to make spreadsheet edits. When the user is done making edits, the edits are posted back to the original grid in the form.

In the spreadsheet editor, users can do the following:

- Copy and paste values from another spreadsheet to the grid (and vice versa)
- Cut, copy, and paste values within the grid
- Drag to copy and fill values across multiple cells
- Use formulas to calculate values (though only the result value will be posted back to the grid in the form; the formula is not preserved)

Requirements and limitations

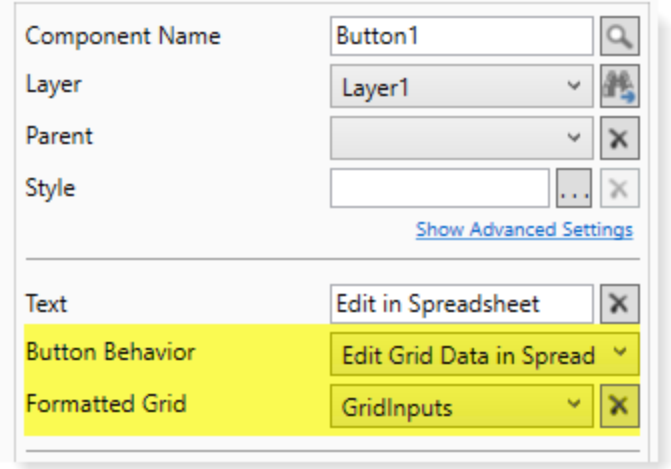
- You can enable spreadsheet editing by using a Button component that is configured to use the **Edit Grid Data in Spreadsheet** button behavior. Button tags for Formatted Grid components can also use this behavior.

- The **Edit Grid Data in Spreadsheet** button does *not* trigger the form update cycle. The current grid values are read from the form (including any unsubmitted changes) and then rendered in the spreadsheet editor. When the user closes the editor, the display of the form grid is updated for changed values, but these changed values are not submitted back to the source file (even if the grid is configured to auto-submit).
- When the grid contents are displayed in the spreadsheet editor, the contents start at A1 and continue to the last cell displayed in the grid. Blank cells in the grid display as blank cells in the spreadsheet editor, including entirely blank rows and columns.
- Only unlocked cells and cells with interactive controls are eligible to be edited in the grid. However, when the grid is opened in the spreadsheet editor, any cell can be changed in the spreadsheet editor. The spreadsheet editor does not provide any indication of which cells are eligible to be edited. When the user closes the spreadsheet editor to submit the changes back to the grid, only the changes to eligible cells will be posted back to the grid—any other changes will be lost.
- Interactive controls such as drop-down lists and check boxes do not display in the spreadsheet editor. Instead, the current input or selected value is displayed as regular text and can be edited. For example, if a cell contains a Checkbox tag in the source file, and the interactive check box is currently selected in the form grid, that cell displays as 1 in the spreadsheet editor.
- The spreadsheet editor does not display any formatting other than number formatting. Shading, bold, underline, etc. do not display. Special display elements such as symbols and sparkline charts do not display.
- The user cannot add new rows or columns using the spreadsheet editor. Only the existing grid contents can be edited. If any content is added in a new row or column, it is lost when the user closes the spreadsheet editor.

Setting up a button for spreadsheet editing

To configure a button to launch the spreadsheet editor, add the Button component to the Axiom form canvas and then configure the properties as follows:

- Set **Button Behavior** to **Edit Grid Data in Spreadsheet**.
- Set **Formatted Grid** to the name of the Formatted Grid component that you want to open in the spreadsheet editor. The Formatted Grid property is only visible once the button behavior has been set to Edit Grid Data in Spreadsheet.

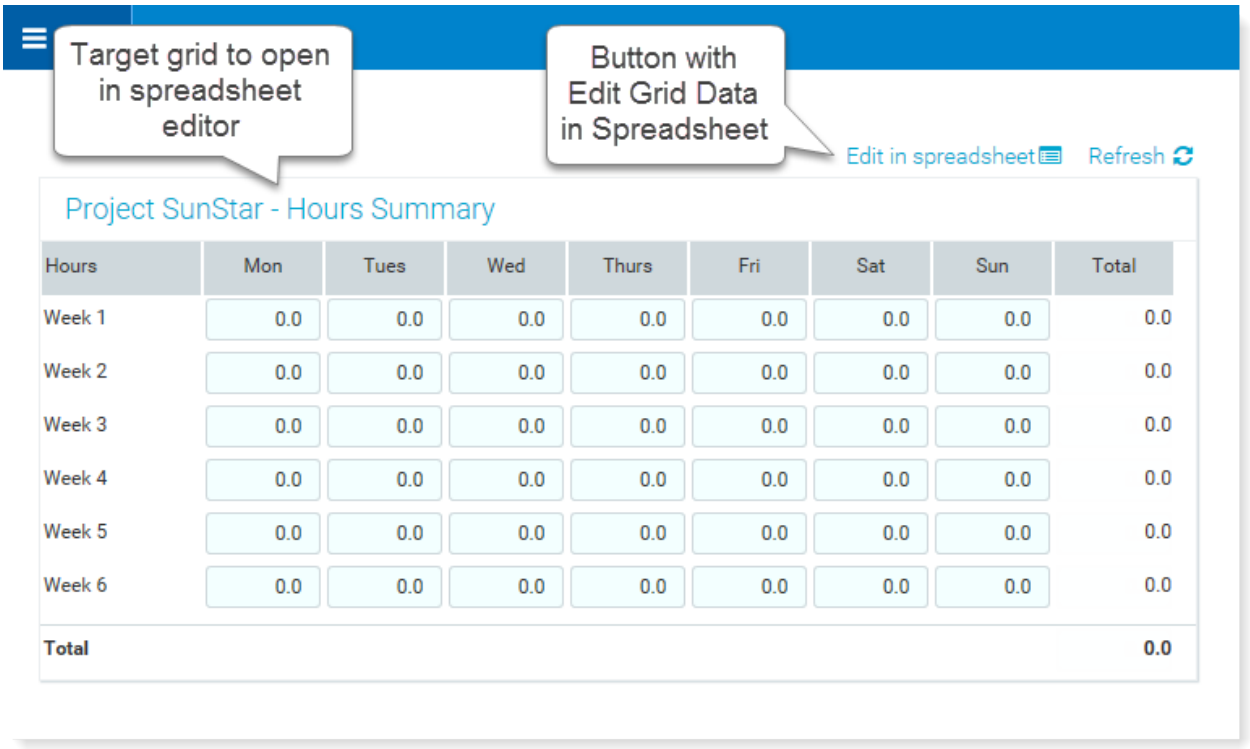


All other visible button properties can be set as desired. Note that it is not possible to run a command or to save data to the database when using this button behavior.

Button tags for Formatted Grids can also use this button behavior. You may want to put the button in the grid itself instead of using a separate Button component. The Tag Editor and Data Source Assistant allow selecting this button behavior and specifying the Formatted Grid component to open. However, in this case you must manually type the name of the desired Formatted Grid component—the tag helpers do not have a drop-down list that allows you to choose component names.

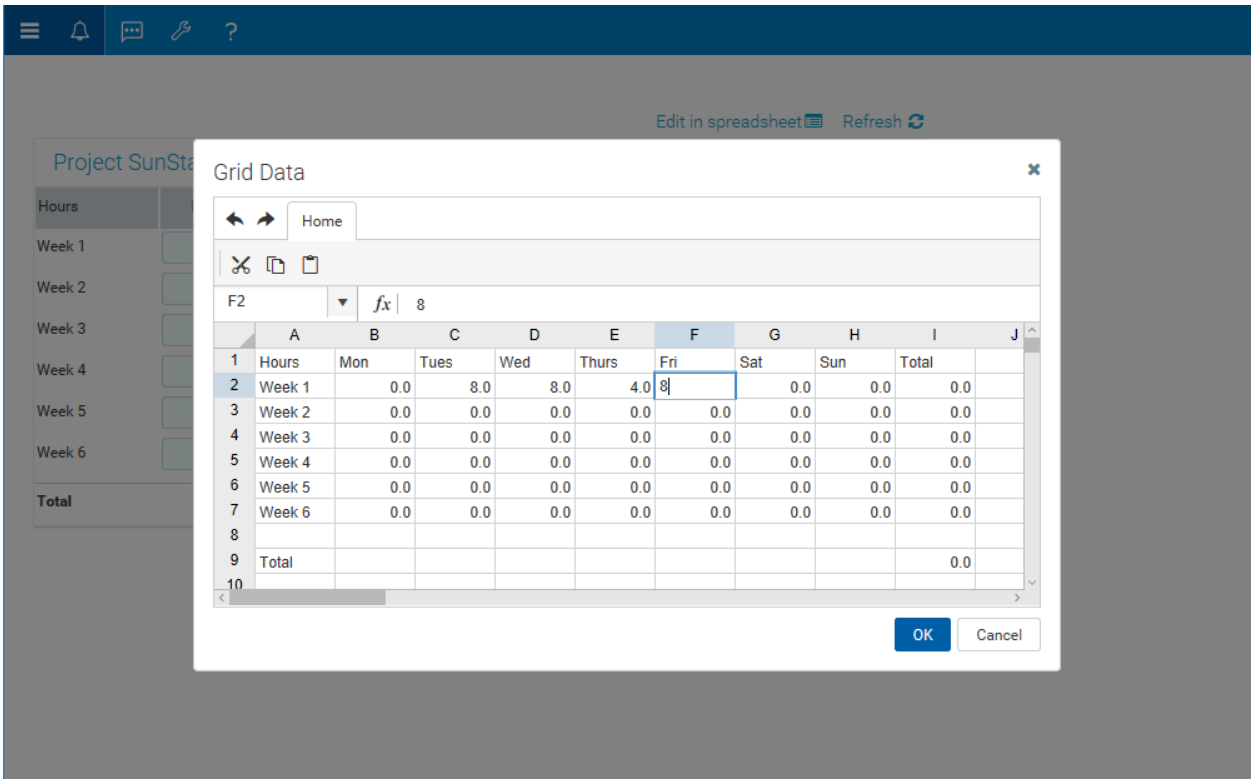
Example

The following example form contains a Formatted Grid component where users can enter hours spent on a project. The form contains a button that is set up with the **Edit Grid Data in Spreadsheet** behavior, where the target is the hours grid.



Example grid to open in editor

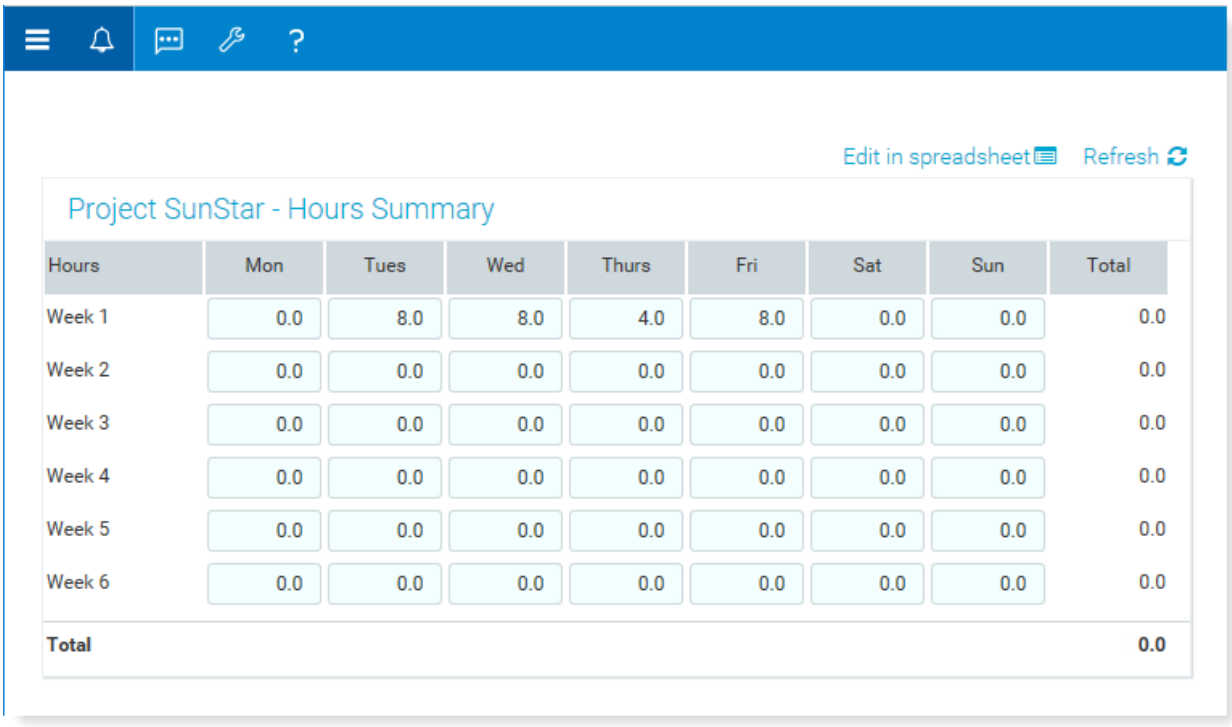
When the user clicks the button, the contents of the target form are opened in the spreadsheet-style editor, in a modal dialog. The user can edit cells, copy/paste from Excel, or drag to copy or fill. In this example, the user has entered some time into the initial week of the project.



User modifies values in the spreadsheet editor

NOTE: The spreadsheet editor does not provide any indication of which cells are editable, and does not prevent edits in locked cells. If an edit is made in a locked cell, that edit is ignored when changes are posted back to the grid in the form.

When the user clicks OK in the spreadsheet editor, the editor is closed and the changes are posted back to the Formatted Grid component. In this example, the hours added to Week 1 are now reflected in the grid.



Hours	Mon	Tues	Wed	Thurs	Fri	Sat	Sun	Total
Week 1	0.0	8.0	8.0	4.0	8.0	0.0	0.0	0.0
Week 2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week 4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Week 6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Total								**0.0**

Changed values from the spreadsheet editor are posted to the grid

Notice that the totals in the last column have not updated for the added hours. This is because using the spreadsheet editor does not trigger a form update. The new values have not yet been submitted back to the source file, so the formula has not updated for the new values. In this example, a Refresh button has been provided on the form to update the grid for the new values.

Exporting Formatted Grid contents to a spreadsheet

You can set up a form so that users can export the contents of a Formatted Grid component to a spreadsheet file. This might be done as a substitute for printing the form, or to allow users to perform further manipulations of the data within a spreadsheet.

Requirements and limitations

- You can enable export for a grid by using a Button component that is configured with the **Export Grid** command. Only one grid can be exported per command button.
- The form update and refresh process still occurs when the export button is used, however, the Axiom form itself is not updated when the process is complete. The purpose of the update is simply to prepare the grid for export. However, you should be aware that the refresh is occurring and disable Axiom queries as necessary so that they do not change the grid contents unintentionally. The **Is Excel Export** setting on the Form Control Sheet is automatically set to On when the process begins, and then set back to Off when the process is complete.

- Only the contents of the Grid data source are exported. Formulas are converted to values. Configuration details set on the Formatted Grid component itself are ignored, such as the title bar and overall component border.
- For thematic grids, no formatting is applied to the exported data, other than honoring number formats. If the grid is a spreadsheet-formatted grid, then most formatting defined in the spreadsheet is preserved, such as fonts and colors. However, row heights and column widths are not applied to the exported data.
- Once the grid contents have been exported to a spreadsheet file, that file is downloaded to the browser. The file name is the name of the formatted grid. The browser prompts the user to save or open the file. The specific behavior of this download prompt depends on the browser used.
- The export is not supported for use on iPad or other tablets.

If the grid contains content tags, these tags will be converted to values as follows:

Tag	Export Result
Button	Button text
Checkbox	Checked status (True or False)
DatePicker	Selected date as Excel date serial number (you must format the target cell as a date in order to show this value as a date in the exported data)
Format	Display text
Href	Display text (not a clickable hyperlink)
Select	Label for selected value
Sparkline	Blank cell
Symbol	Not supported for export; Symbol tag is returned
TextArea	Inputted text

If desired, you can modify the grid for export so that unsupported tags are not included. See Modifying the grid for export.

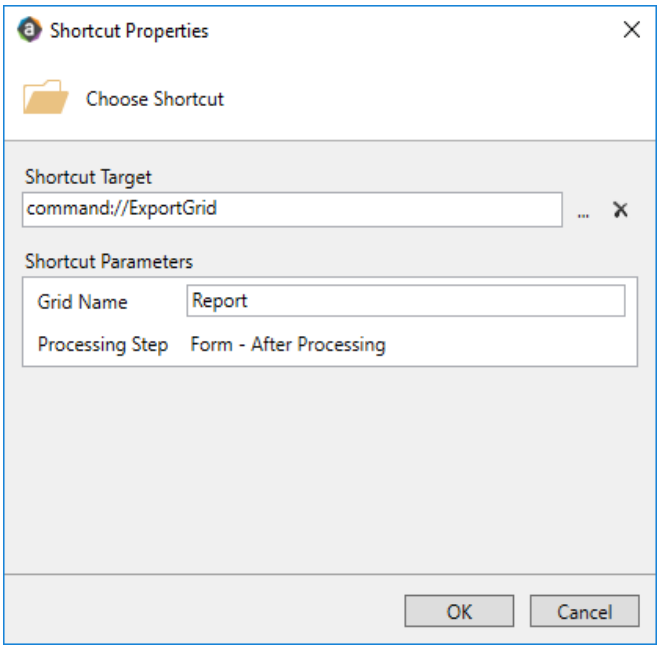
Setting up a button for grid export

To export the contents of a formatted grid to a spreadsheet, you must use a Button component that is configured with the **Export Grid** command. When the user clicks the button, the contents of the specified formatted grid will be exported.

To start off, add the Button component to the Axiom form canvas and then configure the properties as desired. The button text should be defined as something like "Export Grid to Spreadsheet". You can then configure the **Command** for the component as follows:

1. In the Button component properties, click the **[...]** button to the right of the **Command** box.
2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.
3. In the **Axiom Explorer** dialog, navigate to the **Command Library**. Select the **Export Grid** command, then click **Open**.

   The Export Grid command is now listed as the shortcut target, and the relevant shortcut parameters are now available.
4. In the shortcut parameters, type the component name of the grid that you want to export, then click **OK** to close the Shortcut Properties dialog.



Example Shortcut Properties dialog

The button can now be used to export the contents of the specified formatted grid.

Modifying the grid for export

If desired, you can modify the grid before its contents are exported, by using dynamic formulas that reference the **Is Excel Export** setting on the Form Control Sheet. For example if the grid contains a column with Symbol tags, then you may want to omit that column from the export, or change the contents of those cells.

When the button with the Export Grid command is clicked, the following occurs:

- The **Is Excel Export** setting is toggled to **On**.
- Updated values from the form are submitted back to the source file, and the normal refresh process occurs. This allows the grid to dynamically adjust for the export, by using formulas that reference the **Is Excel Export** setting. Once the refresh is complete, the grid is exported to a spreadsheet file.
- The **Is Excel Export** setting is toggled back to **Off**, and the file is calculated so that any formulas referencing the setting will adjust as appropriate. Ideally, this restores the file back to the state it was in before the export process began.
- The Axiom form is *not* updated at the end of the process. This is because the purpose of the refresh is to prepare the grid for export, not to impact the display of the current form.

For example, if you want to hide a column that contains symbols for purposes of the export, you could use a formula such as the following to define the column tag:

```
=If(Control_Form!D23="On","NoColumn","[Column]")
```

Where Control_Form!D23 is the cell address of the **Is Excel Export** setting on the Form Control Sheet. The column will be visible normally, but when exporting grid contents the column will be hidden.

The **Is Excel Export** setting should be used instead of the **Triggering Component** to impact the grid contents, because the export setting will be disabled at the end of the process, whereas the triggering component stays set until another component triggers an update. Using the triggering component can cause the grid contents in the source file to become out of sync with the grid contents presented in the Axiom form.

Using Feature Controls

The feature controls components provide access to pre-built features or accelerate the creation of forms. In the Form Designer, these components are available in the **Feature Controls** section along the left-hand side of the screen.

- Announcements: Users can view and manage announcements.
- Dialog Panel: Places a set of pre-configured components on the form canvas, to be used as a dialog that users can open from within the form.
- Embedded Form: Display another form embedded within the current form.
- Form Help: Users can click a help icon to open a panel that displays custom help content for the current form.
- Menu: Users can select items from a menu to change the content shown in the current form, or to open web content in a new window.
- Process Summary: Users can view new and important process tasks, and access tools to manage these tasks.
- Titled Panel: Places a set of pre-configured components on the form canvas, to be used as a template for further design of a titled form.
- Wizard Panel: Users can move through a defined set of steps and complete a configured action.

Announcements component

The Announcements component can be used to display and manage announcements to users. This component provides a self-contained solution for announcements that does not require any other setup.

Announcements are typically displayed in Axiom Software Home file. You might have a single Home file for all users, or multiple Home pages for different user roles or different products within Axiom Software. You can add an Announcement component to any form where you want to display announcements.

All Announcements components in your system use the same announcements repository stored in the Axiom Software database. However, each component can be optionally configured to only display announcements that belong to certain announcement categories. The list of announcement categories is user-definable. Using categories, you can display different announcements to different audiences.

For more information about using the Announcements component, see Displaying announcements in Axiom forms.

Component properties

You can define the following properties for an Announcements component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Title Text	The title text for the component. This text displays in the title bar for the component within the Axiom form, if the title bar is enabled. If the title bar is disabled, then this text does not display at all in the form.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled. For the Announcement component, the title bar must be enabled if you want users with the appropriate permissions to be able to add new announcements. The plus icon that opens the **Add Announcement** dialog is only present if the title bar is enabled.
Show All Announcements	Specifies whether all announcements display in the component. By default, this is disabled, which means that only current announcements display in the component. If this option is enabled, then all announcements display in the component, including: • Announcements that have not yet started (Start Date is in the future) • Announcements that have expired (Expire Date has been reached) This option is intended for use when creating an "announcements management" form, so that users with the appropriate permissions can view and edit all announcements (past and future), and delete announcements that are no longer needed.

Item	Description
Default Message	A message to display in the component when there are no active announcements. For example: "No announcements available."
Show Categories	Specifies whether category labels display in the component. If enabled, then announcements are grouped under category labels (the category Display Text), so that users can see which category each announcement belongs to. This option must be enabled if you want to filter the component display by specific categories. **NOTE:** This option is ignored if **Show All Announcements** is enabled. Announcements for all categories will display in this case, with the category labels.
Limit Categories To	Specifies whether the component is limited to only showing announcements for certain categories. • If you want the component to display announcements for all categories, then do not select any categories. • If you want the component to only display announcements for certain categories, then select the check boxes for the categories that you want to display. The selected categories are written the Form Control Sheet in a comma-separated list, using the category name (not the display name). If the display name is later changed, the component will still be filtered by the category. To filter by categories, the category names must already be defined. If the categories do not already exist, then you must view the file as a form and use the Announcement component to create the categories, then go back to the component properties and select the categories that you want to show in the component. For more information on managing announcement categories, see Managing announcements. Categories are loaded into the Form Designer when the file is opened. If you add or remove a category while the file is open, you must close and reopen the file in order to see the change. **NOTE:** This option is ignored if **Show All Announcements** is enabled. Announcements for all categories will display in this case, with the category labels.

Item	Description
Collapse Height	Specifies whether the component automatically collapses in height if the configured component height is greater than the content to be shown in the component. You should leave this option disabled if you want the component height to always be the same, no matter how much content is available to display. If the content exceeds the component height, the component will have a vertical scroll bar. If the content does not fill the component, then there will be blank space between the final row of content and the bottom edge of the component. If you enable this option, then when the content does not fill the component, the component will auto-shrink to fit the content instead of maintaining its configured height. The behavior if the content exceeds the component height is the same (vertical scroll bar).

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Announcement components. Only the generic styles are available. Most announcement styling is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Dialog Panel component

Using the Dialog Panel component, you can display a dialog in an Axiom form. When the dialog is launched, it "takes over" the current form, so that the user must interact with the dialog before they can go back to using the form. The content of the dialog is entirely customizable.

Defining a dialog panel is a multi-step process that requires the following:

- Placement and configuration of a Dialog Panel component (and its child components) on the Axiom form canvas.
- Placement and configuration of the dialog contents, using other Axiom form components as needed.
- Placement and configuration of a separate Button component that uses the **Show Dialog Panel** button behavior. This button is how users open the dialog from within the form.

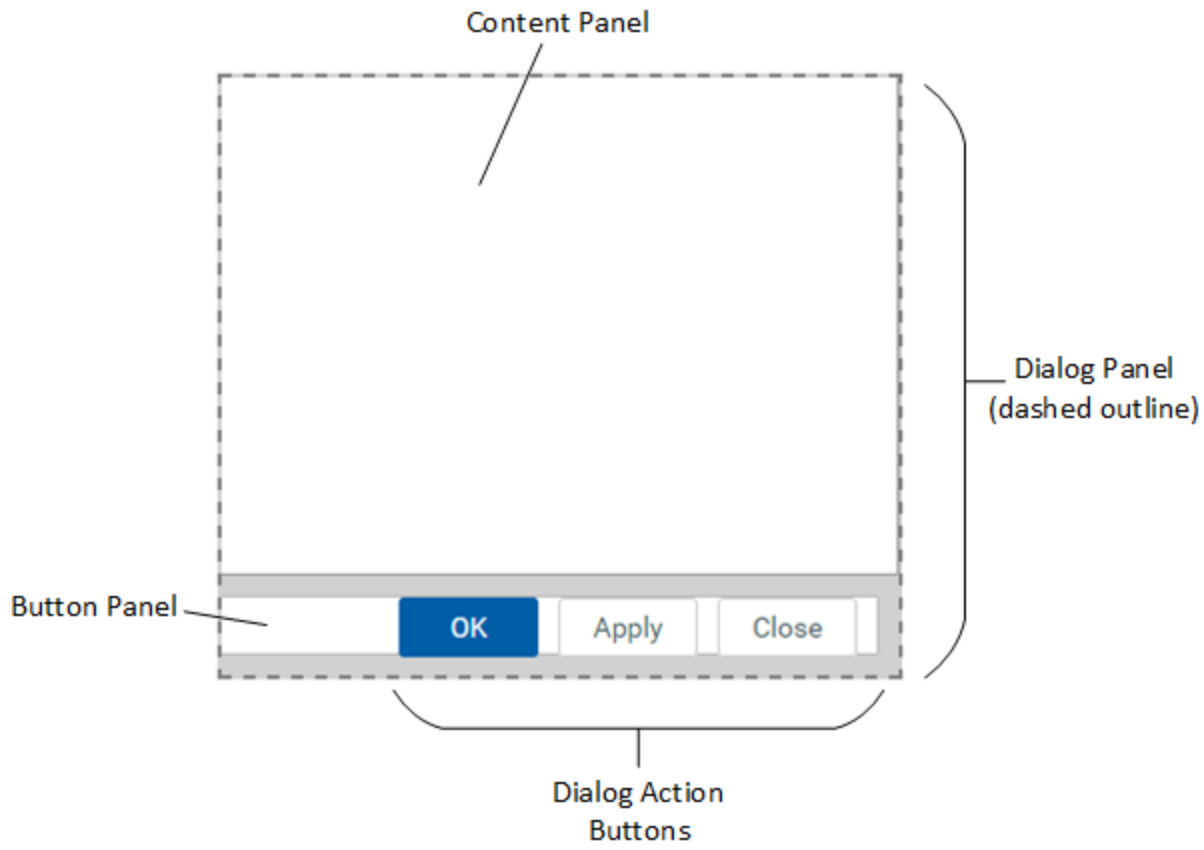
The Dialog Panel component is a composite component. All of the following components will be added to the form when you drag and drop a Dialog Panel component on to the form canvas:

- A parent Dialog Panel component that is used to define the overall size of the dialog, and to position and group the other child components.
- A child Panel component to contain the contents of the dialog (*content panel*).
- A child Panel component to contain and position the action buttons for the dialog (*button panel*).
- Three Button components to perform the dialog actions of OK, Apply, and Close.

NOTE: The Dialog Panel component is only intended for use within an Axiom form—where the primary "application" is the form web page, and you want to present a dialog within that web page. The Dialog Panel component is *not* intended for use when creating a form for use as a custom dialog in the Desktop Client, such as an Add File Form or a RefreshDialog refresh form. When creating a form dialog for use in the Desktop Client, the entire form is the dialog. For more information, see Custom Dialogs and Task Panes in the Desktop Client.

Dialog Panel overview

When you drag and drop a Dialog Panel component on to the canvas, you get a composite component like the following:



For purposes of differentiating the various components in this diagram, borders have been added to the content and button panels, and a background color has been added to the parent dialog panel. On the canvas, all of the panels are white and do not have any borders. (The Dialog Panel itself displays on the canvas with a dashed border as shown here, but this is only so that you can find it on the canvas—it does not render on the component.)

Notice that no title bar shows on the dialog in the Form Designer. When the dialog is opened in the rendered form, it will have a typical dialog title bar that includes title text and a close icon in the top right corner.

Sizing and positioning the Dialog Panel component in the form

Before dragging and dropping the Dialog Panel component on the canvas, it is a good idea to first create a new layer to hold the panel. Make that layer the active layer, and then drag and drop the component on the canvas. This ensures that all of the composite components will belong to the same layer. If you want to change the layer later, you will have to manually update all composite components to move them to the new layer.

The layer is not required to manage the dialog behavior; it is simply convenience for the file designer. You can show the layer in the Form Designer when you want to work on the dialog, and hide the layer when you want to work on the rest of the form contents.

It is not necessary to dynamically configure the visibility of the Dialog Panel component or the layer that it is on. In the form, the dialog only becomes visible when it is launched by a Show Dialog Panel button. Once the dialog becomes visible, it is only closed (hidden) by a Dialog Panel Action button. Similarly, it is not necessary to manually "fade out" the main form contents when the dialog is visible; the form contents will be masked by a translucent gray layer automatically.

Note the following when sizing and positioning the Dialog Panel component:

- It does not matter where the Dialog Panel component is placed on the form canvas. When the dialog is opened, it is always centered in the middle of the visible form window.
- The size of the Dialog Panel component determines the size of the dialog when it is opened in the form. The dialog cannot be resized by form users.
- To define the size of the dialog, you can resize the Dialog Panel component on the canvas or by manually editing the width and height in the advanced component settings. The child panels will automatically adjust to fit the new size. The width and height must be set using a fixed number of pixels; dynamic sizes are not supported.

It can be difficult to select the Dialog Panel component on the canvas. You can use the component search feature to find and select the Dialog Panel component, or you can try clicking on the very bottom of the dialog outline on the canvas. Another tip is to click on the content panel and then quickly click again to move the selection to the component directly underneath (which in this case is the Dialog Panel). Check the Component Name property to verify that you have selected the Dialog Panel component before attempting to move or resize the dialog.

Component properties

When you drag a Dialog Panel on the Axiom form canvas, the following preconfigured components are added to the form. See the previous section for an example of how these components are positioned.

Dialog Panel

The Dialog Panel component is named **DialogPanel1** by default (subsequent components will increment the number). The Dialog Panel component is a unique component type for purposes of managing the behavior of opening and closing the dialog.

For its component properties, the Dialog Panel component has the same properties as the regular Panel component, with the following exceptions:

- **Title Text**: This property defines the window title text for the dialog. Note that this window title bar is not visible in the Form Designer, but it will display when the dialog is rendered in the form.
- **Show Title Bar**: This property does not apply to Dialog Panel components. The rendered dialog uses a standard window title bar that includes the defined title text.

Content Panel

The content panel is a regular Panel component named **DialogPanel1_Content** by default. The content panel is preconfigured to automatically fill the majority of the parent Dialog Panel component, with the

exception of the button area along the bottom. The content panel is intended to hold the contents of the dialog.

It is not recommended to manually move or resize the content panel. By default, **Lock Layout** is enabled for this panel so that it cannot be accidentally adjusted while working on the canvas. The content panel will automatically resize as the parent Dialog Panel is resized.

The dialog contents are entirely up to the form designer. To define the dialog contents, add components to the content panel as desired. The content panel can use any form component, such as labels, images, grids, text boxes, combo boxes, and more. Keep in mind the following:

- If the dialog contents contain an auto-submit component or a regular Button component, the dialog will refresh to show changes but the main form underneath does not refresh until the OK button or the Apply button is clicked.
- Clicking the Close button prevents any unsubmitted changes in the dialog from being submitted, but it does not revert any changes that were already submitted.

Button Panel

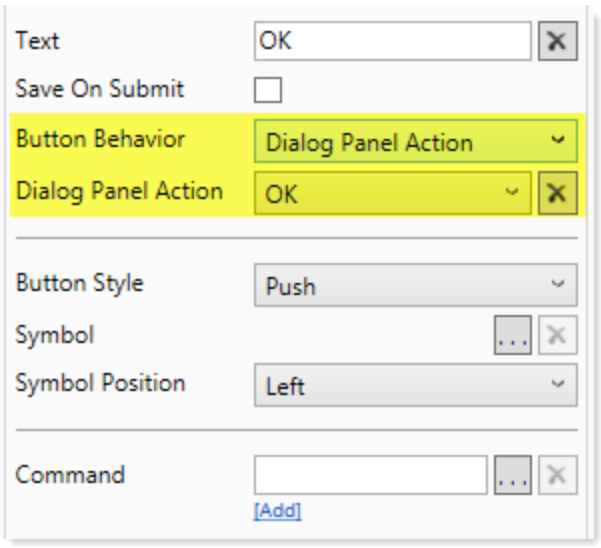
The button panel is a regular Panel component named **DialogPanel1_Buttons** by default. The button panel is preconfigured to fill the bottom of the Dialog Panel and to position the child Button components in the lower right area of the dialog.

It is not recommended to manually move or resize the button panel. By default, **Lock Layout** is enabled for this panel so that it cannot be accidentally adjusted while working on the canvas. The button panel will automatically resize as the parent Dialog Panel is resized (note that you may need to refresh the canvas after resizing the parent panel).

In order to automatically position the buttons, the button panel uses a flow layout with a right-to-left direction.

Action Buttons

Three regular Button components are present in the button panel by default, to control the dialog actions. All three buttons are push-style buttons that are preconfigured to use the **Dialog Panel Action** button behavior.



The text of the button corresponds to the Dialog Panel Action it is configured to perform. The following actions are available:

Button	Description
Close	The Close action closes the dialog without performing a form update, and returns the user to the main form. If the dialog contents contain any interactive components with changes that have not yet been submitted back to the source file, these changes will be lost. By default, this button is named **DialogPanel1_Button1**. **NOTES:** • It is not recommended to rename the button text to Cancel, as "cancel" implies a rolling back of changes that does not occur when the dialog is closed. If any changes have been submitted or data saved while the dialog is opened, these actions cannot be undone. • Clicking the Close button does not cause the component name to be written to the **Triggering Component** field, because the only action that occurs is to close the dialog. The triggering component will continue to be recorded as the last component that triggered a form update (for example, likely the button that was used to open the dialog).
Apply	The Apply action triggers a full form update, including a save-to-database if **Save on Submit** is enabled for the button. The dialog remains open, and the main form is refreshed "underneath" the dialog. By default, this button is named **DialogPanel1_Button2**.

Button	Description
OK	The OK action is the same as the Apply action, except that the dialog is closed at the end of the process and the user is returned to the main Axiom form. By default, this button is named **DialogPanel1_Button3**, and it uses the **primary** style to designate it as the primary action button.

All three buttons are child components of the button panel, and have **Lock Layout** enabled by default. The buttons do not have defined positions because their parent panel uses flow layout. The buttons automatically flow from right to left within the button panel.

The action buttons are designed to be customized as follows:

- **Save data:** You can optionally enable **Save on Submit** for the OK button and the Apply button, if you intend the dialog to perform a save-to-database. Save on Submit cannot be enabled for the Close button, as the Close action only closes the dialog and does not trigger the form update cycle.
- **Perform commands:** You can optionally add commands to the Apply button and/or the OK button, to perform additional actions when the button is clicked. For example, you could process action codes, run a scheduler job, execute data lookups, or add a new record to an identity table. Commands cannot be added to the Close button, as the Close action only closes the dialog and does not trigger the form update cycle.

  Keep in mind that some form commands do not apply to the dialog context and will either be ignored or will not work as expected: About Axiom Software, Apply Form State, Close Dialog (this is only for use with custom form dialogs in the Desktop Client), Export Grid, and Forms Logout.
- **Remove unneeded buttons:** You can delete the Apply button if the dialog does not need to support the ability to update and save without closing the dialog. Once the Apply button is deleted, the OK button will automatically re-position itself due to the flow layout behavior.

  You can delete both the Apply button and the Close button if the dialog is only being used to display information to the user, and no action needs to be taken other than dismissing the dialog when the user is finished reading it. It is recommended to use the OK action for this, because it will cause the display of the main form to be refreshed for any changes that were submitted when the dialog was launched using the Show Dialog Panel button (see the following section). You can optionally rename the text of the OK button to Close for this use case.
- **Customize text and size:** You can change the text or sizes of the buttons. By default, the buttons are all sized to 80px width and 30px height. If you change the text on a button, you may need to adjust the width larger to fit. Because Lock Layout is enabled for the buttons by default, you must adjust the width manually in the advanced component properties.

If needed, you can add other Button components that do not use the Dialog Panel Action behavior to the button panel or the content panel. You might do this if you need the user to be able to perform an action independently from the dialog actions. Keep in mind that the main form contents underneath the

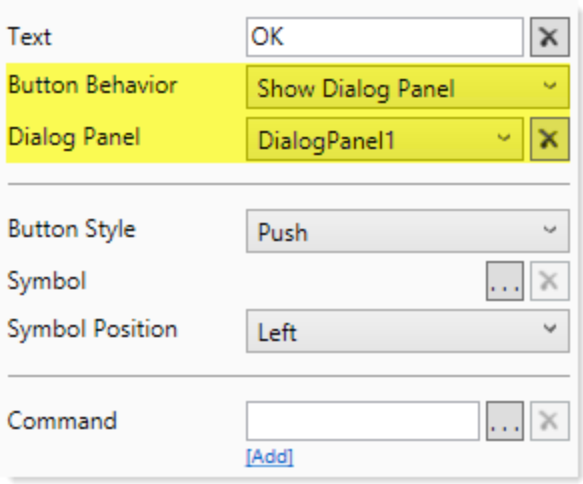
dialog will not refresh to show changes when a regular button is used; only the dialog contents will refresh. The main form contents are only refreshed when an OK or Apply button is clicked.

NOTE: Nested Dialog Panel components are not supported. You cannot use a button to launch a second Dialog Panel component while the first dialog remains open.

Configuring a button to open a Dialog Panel

In order for users to open the dialog created by the Dialog Panel component, the form must contain a button that is configured to use the **Show Dialog Panel** button behavior. You must separately add this button to your form; placing a Dialog Panel component in your form does not add this button. The Show Dialog Panel button behavior is available for both Button components and Button tags for Formatted Grid components.

Once this behavior is selected, use the **Dialog Panel** property to specify the Dialog Panel component to open when the button is clicked. You can select any Dialog Panel component that is currently defined in the form, using the component name.



NOTE: Dialog panels can also be opened by using the Show Form Dialog Panel command. This command is primarily intended for use with components that do not have access to button behaviors, such as the KPI Panel. Although it is possible to use the command with a Button component and use Command behavior instead of Show Dialog Panel behavior, other button options that are available with Command behavior are still not supported when opening a dialog panel, such as **Save on Submit**.

When the button is clicked, the specified Dialog Panel component is opened as a modal dialog over the main form contents. The main form contents are masked in translucent gray.

The dialog remains open until it is closed by using one of the following:

- The X in the right-hand corner of the dialog title bar. This performs the same action as the Close action for buttons.
- A **Dialog Panel Action** button configured to use the Close action.
- A **Dialog Panel Action** button configured to use the OK action.

While the dialog is opened, users cannot interact with the main form contents. Only the dialog contents are active.

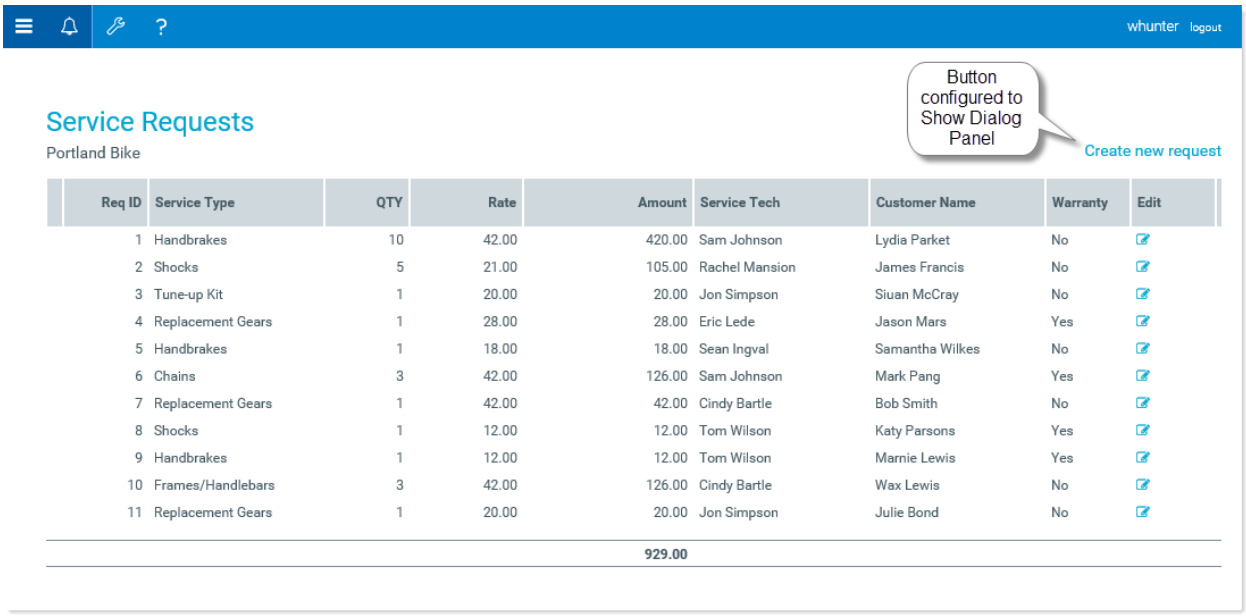
Form update behavior for Show Dialog Panel buttons

When a Show Dialog Panel button is clicked, the form update cycle proceeds as normal up through the **After AQ Refresh** processing step (see Axiom form update process for more information). After that point, the specified Dialog Panel component is opened as a dialog and the remainder of the form update cycle is aborted. This means:

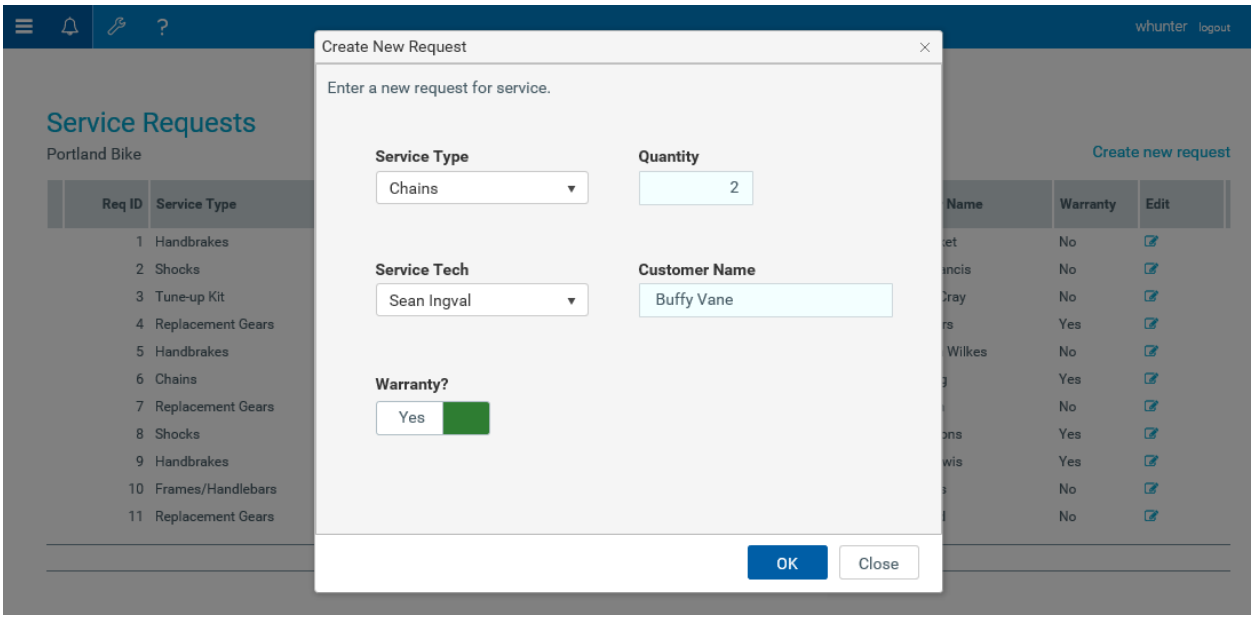
- The save-to-database portion of the cycle is skipped. The Show Dialog Panel button cannot be configured to save on submit. Any Axiom queries and commands that are configured to execute after saving data will not be run.
- The display of the main form is not refreshed. Instead, the specified Dialog Panel component opens and overlays the form. The contents of the dialog can be impacted by submitted values and the by results of the data refresh, but the display of the main form will not be refreshed until either the OK button or the Apply button is clicked on the dialog. If these buttons are not used and instead the dialog is closed via the Close button (or the X button in the title bar), then the display of the main form will not be refreshed.
- You can assign commands to a Show Dialog Panel button and execute those commands as part of opening the dialog. These commands must be run by the **After AQ Refresh** processing step or earlier.

Dialog Panel behavior

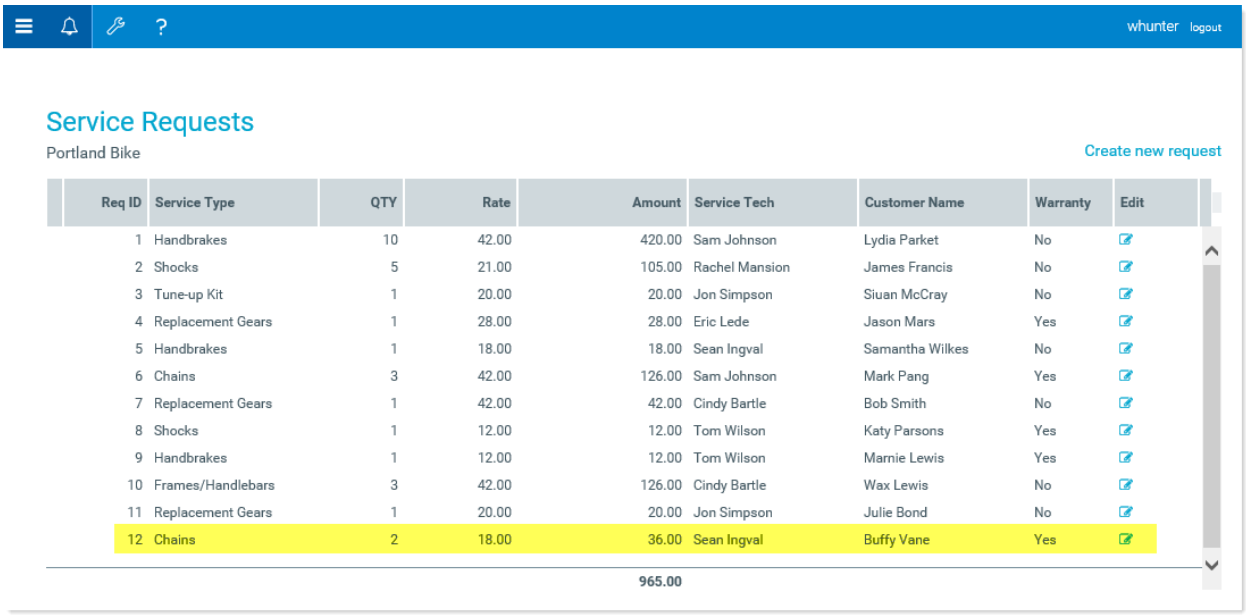
The following example shows how a Dialog Panel can be used within a form. This form displays a list of current service requests for the organization, generated by an Axiom query. If a user wants to create a new service request, they can click the button at the top of the form. This button is configured to use the Show Dialog Panel action.



When the user clicks the button, the Dialog Panel opens. In this case, the contents of the dialog contain the fields necessary to create a new service request. Additionally, the OK button is configured to trigger a save-to-database. When the user completes the fields and then clicks OK, the new request is saved to the database.



When the user returns to the main dialog, the Axiom query for the service requests has been refreshed and the new request now displays in the list.



Embedded Form component

The Embedded Form component displays the contents of an Axiom form embedded within another Axiom form.

The form that contains the Embedded Form component is known as the *parent form*. When you configure the Embedded Form component, you must point it to a target form that will be displayed within the component. This target form is known as the *child form*. When a user views the parent form, the target child form displays as embedded within the parent form. The target child form can always be the same form, or you can use the separate Menu component to allow users to dynamically switch between child forms.

The primary use of this component is to create composite forms, where the form web page that displays to the user is sourced from multiple files instead of a single file. The parent and child forms together make up the composite form to be displayed to the user. Use of composite forms can simplify form creation for complex forms, by separating the form contents into smaller, logical units.

The Embedded Form component can also be used to enable reuse of the same content in multiple forms. For example, imagine that you want to use the same Map View component within two different Axiom forms. Instead of creating the same map configuration within those two Axiom forms, you could instead create a separate Axiom form that contains just the Map View component. You can then embed that Axiom form within each of the Axiom forms where you want to show that map.

NOTES:

- When using the Embedded Form component, the contents of the embedded form do not display in the Form Designer. Instead, the Form Designer displays information about the target form for the Embedded Form component. You must preview the form in order to see how the embedded form is rendered.
- This component requires **Use Web Client Container** to be enabled for the form. By default, the container is enabled for new forms. For more information, see Using the Web Client Container with Axiom forms.

Component properties

You can define the following properties for an Embedded Form component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Title Text	The title text for the component. This text displays in the title bar for the component within the Axiom form, if the title bar is enabled. If the title bar is disabled, then this text does not display at all in the form.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.

Item	Description
Source Menu Component	Specifies the Menu component to associate with the Embedded Form component, so that the Embedded Form component will automatically display the currently selected item in the menu. You can select any Menu component that is currently defined in the form, using the component name. This is the recommended method to link a Menu component and an Embedded Form component. The data source for the Menu component must be populated with file paths to form-enabled documents, so that the defined values are valid for display within the Embedded Form component. When using this approach, the ID for each menu item is automatically added to each child form's generated instance ID. This ensures that each menu item has a unique instance ID, even if multiple menu items use the same target file as the value. This means that you can open the same target file from the menu multiple times and each instance is maintained separately, thereby enabling each instance to be in different states. When you specify a source menu component, the **Form Document** and **Instance Identifier** properties are unnecessary and become hidden in the Form Designer and Form Assistant. If any values are defined for these properties on the Form Control Sheet, they will be ignored by the form.

Item	Description
Form Document	The path to the form-enabled file to display within the Embedded Form component. Click the **[...]** button to browse to the desired form. **IMPORTANT:** This property only applies if you are *not* using the Embedded Form component with a Menu component. If you are using a Menu component, then complete the **Source Menu Component** property instead. Once a source menu component has been selected, the Form Document property becomes hidden in the Form Designer and Form Assistant, and any existing value is ignored by the form. Generally speaking, the Form Document setting is only for use in cases where the target embedded form is "fixed" and the component will always display that form. **NOTES:** • Users must have security permission to the target form in order to see it rendered as an embedded form. Permission to the parent form is not sufficient. • The next time you open this file after saving, the path to the form will be automatically converted into a system-managed document shortcut (you can tell the difference by the presence of a `_tid` parameter on the end of the shortcut). This is to make the file reference "repairable" in cases where the file is renamed or moved. Note that if the path is a result of a formula instead of directly within the cell, then the conversion will not occur and the file reference will not be repairable. • Files created in earlier versions may be using the following syntax to automatically use the currently selected value of a Menu component: `[MenuComponentName.SelectedValue]`. Although this syntax will still work, it is strongly recommended to convert the component to use the **Source Menu Component** instead.
Instance Identifier	A unique ID to add to the child form's automatically generated instance ID. Generally speaking, this property should only be used if the form contains multiple Embedded Form components, and two or more of those components use the same target form document. For more information, see Using multiple Embedded Form components in a single form. **NOTE:** This property does not apply if you are using a Source Menu Component.

Item	Description
Save on Parent Submit	Specifies whether a save-to-database is executed in the embedded child form when the parent form is updated. By default, this is disabled. When the parent form is updated, the child form will not attempt to execute a save-to-database (unless the triggering component in the parent form has **Save on Submit** enabled). If enabled, then whenever the parent form is updated using any component, Axiom Software will attempt to execute a save-to-database in the child form. This is most often used in conjunction with a Menu component, to save data before moving from one child form to another child form using the menu. For more information, see Saving data from composite forms.
Force Refresh	Specifies whether the embedded child form is updated when the parent form is updated. By default, this is disabled. When an update is triggered in the parent form, the embedded child form is not automatically updated. This means that the child form will not update to reflect changes made in the parent form until the child form is separately updated. If enabled, then the embedded child form will be updated whenever the parent form is updated. The parent form is updated first, followed by the child form. This allows the child form to be updated for changes made in the parent form, such as a change to a shared variable, or a change to a value saved to the database. For more information on form update behavior in composite forms, see Form session and update behavior for composite forms. **NOTES:** • This option is not available if **Refresh Parent Form** is enabled. Since Refresh Parent Form is enabled by default, you must first disable that option in order to make Force Refresh visible in the component properties. • If the Embedded Form component is being used in conjunction with a Menu component, then this option should not be used. Instead, use the `[ForceRefresh]` property of the Menu data source to indicate whether a particular child form should be updated.

Item	Description
Refresh Parent Form	Specifies whether the parent form is updated when the embedded child form is updated. By default, this is enabled. When an update is triggered in the child form, the parent form will also be updated. The child form is updated first, followed by the parent form. This allows the parent form to be updated for changes made in the child form, such as a change to a shared variable, or a change to a value saved to the database. If disabled, then the parent form will not be updated when the child form is updated. This means that the parent form will not update to reflect changes made in the child form until the parent form is separately updated. This should only be disabled if the parent form does not depend on values set by the child form, or if it is not important for the parent form to be immediately updated for those changed values. For more information on form update behavior in composite forms, see Form session and update behavior for composite forms. **NOTE:** This option is not available if **Force Refresh** is enabled.
Overflow	Specifies the behavior if the target form is larger than the container area of the Embedded Form component. Select one of the following: • **Auto** (default): Scroll bars are added to the Embedded Form component only if the target form extends beyond the container area. • **Visible**: The target form is visible beyond the container area. This may cause the target form to interfere with the expected display of other components (for example, to overlap another component). • **Hidden**: Any part of the target form that extends beyond the container area is hidden. This may cause the target form to appear "cut off." • **Scroll**: Scroll bars are always present on the Embedded Form component, regardless of whether they are needed. If the target form extends beyond the container area, the scroll bars are active.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Embedded Form components. Only the generic styles are available. Any styles applied only affect the embedded form container; they do not affect the display of the embedded form itself.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Designing the embedded form

When designing a form to be used as an embedded form, you should keep in mind certain design considerations.

Sizing

The contents of the target form are displayed in actual size within the container area defined by the Embedded Form component (unless **Scale to Fit** is enabled for the embedded form, which is *not* recommended). If the target form is larger than the container area, the behavior is determined by the **Overflow** property.

If you want the target form to fill the Embedded Form component, then the contents of the target form should be configured to dock the width and height. The easiest way to accomplish this is to use a panel that is configured to dock, and then place all of the contents in this panel. You can use percentage sizes for components in the target form so that they adjust to the size of the panel (which is in turn adjusting to the size of the Embedded Form component).

If you use this type of configuration, keep in mind that the contents may not adjust well if the Embedded Form component is sized smaller than the minimum optimal size of the target form. Make sure that the Embedded Form container is sized appropriately to display the target form contents, and to fill the desired space in the parent form. For example, you may want to dock the Embedded Form component within the parent form as well.

NOTE: The canvas size of the target form is ignored if set.

Appearance

The following form-level settings are treated as follows for the target form:

- The skin and theme of the target form are honored. This means that both may be different than the parent form. However, while it makes sense for an embedded form to use a different theme than the parent, it is recommended to use the same skin so that the styling elements in both forms match.

- The Web Client Container does not display on the target form, however, it is recommended to leave it enabled for the target form in case it is required to display certain elements in the form. The container will display on the parent form or not depending on the parent form's configuration.

Content limitations

Generally speaking, embedded forms can use any form feature. Some limitations apply:

- Nested embedded forms are not supported. This means that the target form displayed in an Embedded Form component should not itself contain another Embedded Form component.
- Generally speaking, the form-specific features in the Web Client Container apply to the parent form only. For example, if the Message Stream is present, any comment entered is associated with the parent form, not any embedded child forms. It is not possible to add or view comments for the child forms. Once exception is the Filters panel, which can be used to apply refresh variables for both the parent and child forms. For more information, see Filters panel behavior for composite forms.

Form update behavior

Some special update behaviors apply when a form is embedded within another form. For example:

- By default, triggering an update in the child form does not cause the parent form to be updated, unless **Refresh Parent Form** is enabled for the Embedded Form component.
- By default, triggering an update in the parent form does not cause the child form to be updated, unless one of the following options is used:
    - If you are using the Menu component with the Embedded Form component, you can optionally force the child form to be updated by using the `[ForceRefresh]` property in the Menu data source.
    - If you are not using the Menu component, then you can optionally force the child form to be updated by enabling the **Force Refresh** property for the Embedded Form component. However, you cannot enable both Force Refresh and Refresh Parent Form on the Embedded Form component itself—you must choose one or the other (or neither).
- Triggering a save-to-database in the parent form (Save on Submit) will automatically trigger a save-to-database in the child form, if the child form has an enabled save-to-database process. If you do not want the child form to save when the parent form saves, then you can use formulas to disable the child save when the triggering component is **$ParentForm**.
- If a Formatted Grid component is used within a child form, the grid must be set to **Inline** loading behavior. **Asynchronous** loading behavior is not supported with embedded forms.

You should keep these special behaviors in mind when designing interactive elements within the form, as well as other behaviors that depend on the form update cycle, such as save-to-database. For more information, see Form session and update behavior for composite forms and Saving data from composite forms.

Using an Embedded Form component with a Menu component

Embedded Form components are primarily intended to be used with Menu components. As the user selects items in the menu, the Embedded Form component updates to display the target child form associated with the currently selected menu item. This way, a single Embedded Form component can be used to display any number of child forms within the parent form.

Because the two components are designed to work together, each component has certain settings that only apply to this combined use case. The following is a basic summary of how to configure an Embedded Form component and a Menu component to work together.

1. Place both components on the canvas, and adjust the basic component details such as position, sizing, and component name.
2. On the Embedded Form component, set the **Source Menu Component** to the name of the Menu component. This "links" the Embedded Form component to the Menu component, so that the Embedded Form component will automatically use the currently selected value for the Menu component as the target embedded form.

   When using this approach, the **Form Document** and **Instance Identifier** properties are hidden and no longer apply.
3. In the Menu data source, make sure the `[Value]` column is populated appropriately for use with an Embedded Form component. To be valid for display in the Embedded Form component, the menu value must be a file path to a form-enabled document. The data source can also contain URLs or form-enabled file paths that are configured to open in a new window (using the `[NewWindow]` column) instead of within the Embedded Form component.
4. In the Menu data source, use the `[ForceRefresh]` column to determine if the target form should be updated when it is selected in the menu and displayed in the Embedded Form component. The target form will also be updated if it is the currently visible child form and an update is triggered in the parent form. You should set this to True if the target form depends on values that can be changed in the parent form or in other child forms.
5. Optional. In the Menu data source, you can use the `[DocumentVariables]` column to pass values to the target form when it is selected in the menu and displayed in the Embedded Form component. The target form must use the GetDocumentInfo function to return these passed values and impact the form in some way. Generally speaking, document variables should be used when the values only apply to the target form opened by the Menu item. If the values need to be shared by two or more forms, then shared variables should be used instead.

For more information on the Menu component and configuring its data source, see Menu component . For general information and design concepts for composite forms, see Using composite forms.

Using multiple Embedded Form components in a single form

If desired, you can use multiple Embedded Form components within the same form. For example, you might want to:

- Display multiple child forms within a parent form concurrently. The form could be a dashboard where you want to present three distinct blocks of content within the page. Instead of defining each block using Panel components within the parent form, you can define each block using three separate files and then display those files as embedded within the parent form.
- Display the same child form multiple times within a parent form concurrently. Document variables can be passed to the child form instances so that each instance displays different content.

When using multiple Embedded Form components, the child forms can display independent content, or the child forms can depend on shared variables that are set in the parent. If you are using shared variables, then you must enable **Force Refresh** for the Embedded Form components in order to force the child forms to update when the parent form is updated. Otherwise the child forms will not update when the variable value is changed in the parent form. Note that the child forms cannot depend on shared variables that are set within other child forms, because there is no way to force the other child forms to update when one child form is updated.

If you are using the same child form within multiple Embedded Form components, you must do the following:

- Define a unique **Instance Identifier** for each component. The identifier values can be anything you want, as long as they are unique. The values should not contain $ or @.

  This allows Axiom Software to manage each instance of the child form independently. If unique instance identifiers are not defined, then Axiom Software will render the same instance of the form within each Embedded Form component. Additionally, errors or incomplete rendering may occur.
- Append document variables to the **Form Document** path so that the content shown in each instance of the child form is different. For example, you could have a variable Entity and then pass different entity values to the child form, so that each instance displays the data for a different entity. The child form must use the GetDocumentInfo function to return the variable value and filter the data queries by that value (or impact the form contents in some other way).

To append document variables to the path, use the following syntax:

```
\foldername\filename.xslx?variablename=value&variablename=value
```

For example:

```
\Axiom\SystemFolderName_ReportsLibrary\Forms\current_sales.xlsx?Entity=1
```

This example passes the value 1 for the variable Entity, when this target form is opened within the Embedded Form component.

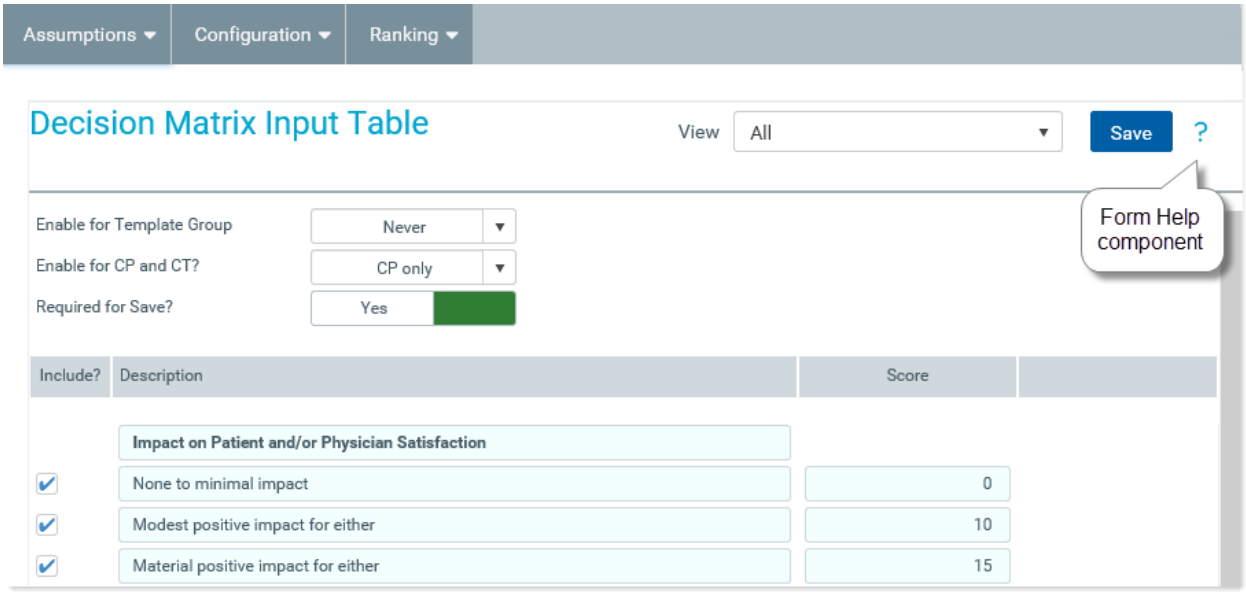
NOTE: Multiple Embedded Form components should only be used when the content needs to display concurrently (side by side within the form). If the child forms can be viewed sequentially, one at a time, then a single Embedded Form component should be used with a Menu component instead. Users can select items in the menu to change which child form currently displays as embedded within the parent.

Form Help component

The Form Help component displays custom help text in an Axiom form. You can use this component to provide information and instructions about the current form to end users.

For example, you may have a form that is used to gather user inputs and save them to the database, and you want to provide users with guidance on how certain fields in the form should be filled out, or give users information on how the inputs will be used after saving. Or the form may contain reporting data and charts, and you want to provide users with more information on the data, or explain certain terms and abbreviations used in the charts.

The Form Help component displays as a question mark icon in the form. When a user clicks on the icon, a panel slides out from the right-hand side of the form to display the help text associated with the component.



Example Form Help component

Displaying custom help for an Axiom form is a two-part process that requires the following:

- Defining a help code and the associated help text that you want to display in the form. This is done separately, using the Form Help Admin page in the Web Client. For more information, see Managing custom help codes. (If a help code already exists for your desired text, you can use that

code in as many forms as needed.)

- Placement and configuration of a Form Help component on the Axiom form canvas. When configuring the component, you specify which help code it should use.

The Form Help component can be used by itself, or in conjunction with the form-level Help Code property. For example, you might use the form-level help code to provide overall help for the form, and then use the Form Help component to provide context-sensitive help for particular sections, fields, or other items in a form.

NOTE: If you want to use a Form Help component in conjunction with a Menu component, to provide help on the various tabs in the menu, then you can specify the help codes directly in the Menu data source. For more information, see Menu component .

Component properties

You can define the following properties for a Form Help component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Help Code	The code that identifies the help text to display in the help panel. For example, you might have a defined code such as Help_Comment_Input that defines help text for a form named Comment_Input. Help codes and their associated text are defined separately, using the Form Help Admin page in the Web Client. For more information, see Managing custom help codes. Currently, it is not possible to look up a help code from within the component properties. You must already know the appropriate code and manually enter it. Or, if you are an administrator, you can go to the Form Help Admin page, search for the appropriate code, and then copy / paste it into the component properties. If the specified code does not match a defined Form Help code in your system, then clicking the help icon will do nothing in the rendered form. No error will occur.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Form Help components. Only the generic styles are available.

Position and size properties

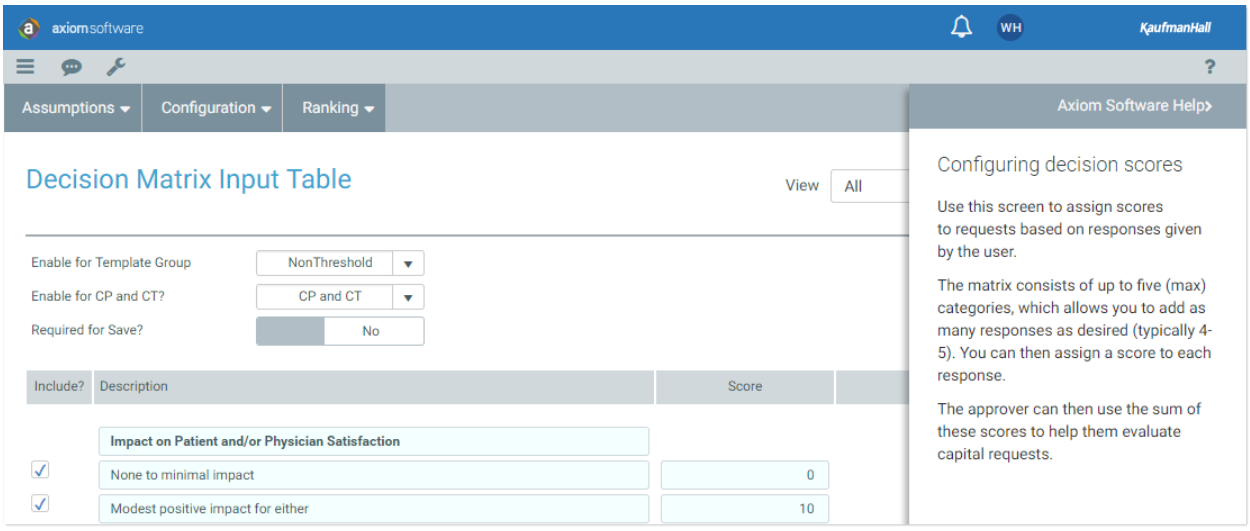
All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

NOTE: The size of the Form Help icon is fixed and does not respond to changes to Width and Height. If you size the component larger than its default size, this will just increase the space taken up by the component; it will not make the icon larger.

Component behavior

The Form Help component displays as a blue question mark icon. It is not possible to change the size or appearance of this icon. The intent is to provide a consistent icon so that users can easily identify the icon in their forms and instantly understand its purpose.

When a user clicks on the icon, a standardized help panel slides out from the right-hand side of the form and displays the help text for the specified help code. The panel overlays the form.



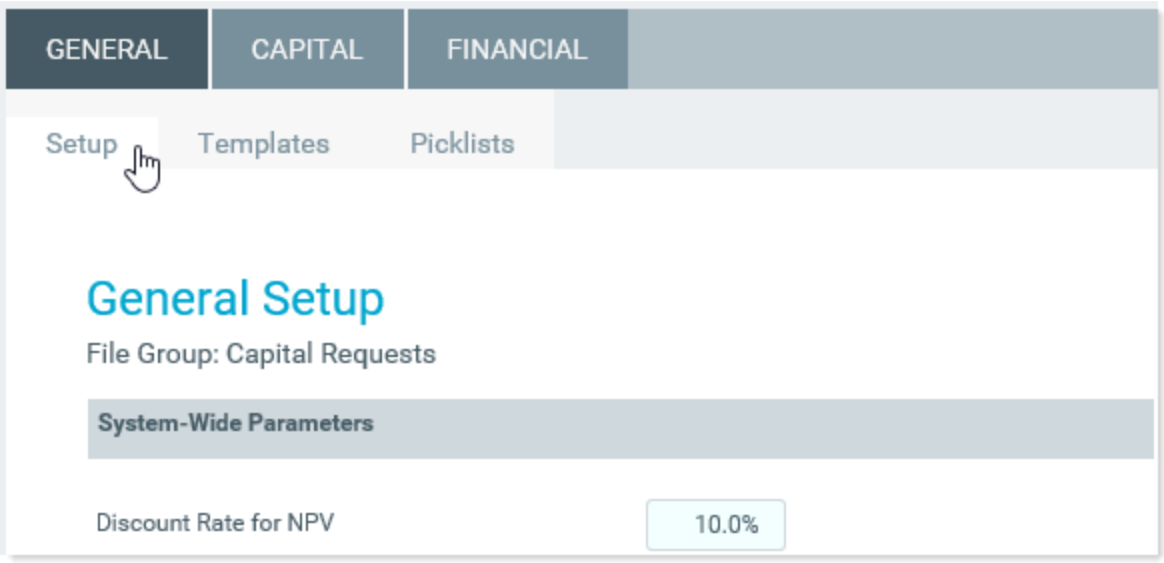
To close the panel, the user can click the question mark icon in the gray task bar, or click inside the form.

Menu component

The Menu component displays a menu control in an Axiom form. Users can select items from the menu to change the current contents shown in the form, or to open content in a new window. For example, menu components can be used to:

- Display different Panel components in an Axiom form, where each item in the menu corresponds to a different panel. As a user selects different items on the menu, the currently visible panel changes as appropriate. (Layers could also be used instead of panels.)
- Display different "child" forms embedded within a "parent" form, where each item in the menu corresponds to a different child form. In this example, the parent form serves as a "frame" that contains the Menu component and an Embedded Form component. As a user selects different items on the menu, the Embedded Form component displays the corresponding child form. This form structure is known as a *composite form*, because the web page that displays to the user is sourced from multiple form documents. For more information on creating composite forms, see Using composite forms.

Menus can be displayed using various styles, such as tabular menus and drop-down menus. The following screenshot shows an example tabular menu with two levels:



Defining a menu is a two-step process that requires the following:

- Creation of a Menu data source in the spreadsheet to define the items to display in the menu.
- Placement and configuration of a Menu component on the Axiom form canvas.

NOTE: This component requires **Use Web Client Container** to be enabled for the form. By default, the Web Client container is enabled for new forms. For more information, see Using the Web Client Container with Axiom forms.

Data source tags

A Menu component must have a defined data source within the file to define the items to display in the menu. The tags for the Menu data source are as follows:

Primary tag

[Menu;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Menu component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[MenuItem]

Each row flagged with this tag defines an item to display in the menu.

Column tags

[ID]

An ID that uniquely identifies each row in the data source. The ID can consist of numbers, text, or a combination of both, as long as it is unique for each row.

When a user selects an item in the menu, the following values for that item are written back to the following fields for the Menu component:

- The value in the `[ID]` column is written back to the **Selected ID** field
- The value in the `[Value]` column is written to the **Selected Value** field

Other components can reference the selected value to change something in the form. For example, if the selected value is the file path to a form, an Embedded Form component can reference that value to change which form it shows.

NOTE: If you are using the menu in conjunction with an Embedded Form component, then the ID values should not contain $ or @.

[Name]

The name of the menu item. This is the text that displays on the menu. The user clicks on the text to select the menu item.

[Value]

The value to apply when the menu item is selected. The value can be anything, though in most cases it will be one of the following:

- A file path to an Axiom form
- A component or layer name
- A URL

The value is either launched directly (for example, if it is a URL) or it is referenced by other components to change something in the form. For more information, see Specifying the value for a menu item.

[ParentID]

Optional. The ID of the parent item for this menu item. The parent ID can be used to create multiple-level menus.

If an item does not have a parent ID, then the item is a top-level menu item. If an item has a parent ID, then the item is shown in a sub-menu underneath the parent item. For more information, see Creating multi-level menus.

[NewWindow]

Optional. Specifies whether the selected menu item is opened in a new window (True/False). The default value is False if omitted or blank.

This option only applies if the value is a URL or a file path to an Axiom form. If True, then the URL or form is opened in a new window instead of within the current window.

When `[NewWindow]` is True, the ID and value of the menu item are *not* written back to the Menu component. The previous ID and value are left as is, so that the form continues to display the current contents while the new contents are opened in a new window.

[Disabled]

Optional. Specifies whether the item is disabled on the menu (True/False). The default value is False if omitted or blank.

If True, then the item continues to display on the menu, but it is grayed out and cannot be selected. Any child menu items underneath the disabled item (if applicable) are disabled as well. This option can be used to dynamically enable or disable a menu item based on a condition.

[Hidden]

Optional. Specifies whether the item displays on the menu (True/False). The default value is False if omitted or blank.

If True, then the item does not display on the menu. Any child menu items underneath the hidden item (if applicable) are hidden as well. This option can be used to dynamically show or hide a menu item based on a condition.

[Tooltip]

Optional. Defines text to display in a tooltip when a user hovers their cursor over the menu item.

[ForceRefresh]

Optional. Specifies whether the target form is updated before it is rendered in the Embedded Form component (True/False). If omitted, the default behavior is False. This column only applies when the value for the menu item is a path to an Axiom form, and the Menu component is being used to switch among various child forms in an Embedded Form component. For more information about this design concept, see Using composite forms.

For example, imagine that when the form first opens, the Embedded Form component shows Form 1. The user then switches to Form 2 via the Menu component, then switches back to Form 1. Since Form 1 has already been opened in the current session, switching back to it at this point does not trigger a form update unless `[ForceRefresh]` is set to True. For more information, see Form session and update behavior for composite forms.

NOTE: If enabled, the currently visible child form is updated whenever the parent form is updated, regardless of whether the update is triggered by using the Menu component. For example, the parent form could have a Button component, or a ComboBox component set to auto-submit. Using either of these components in the parent form causes the child form to be updated.

[DocumentVariables]

Defines one or more document variable/value pairs to pass to the target form. This column only applies when the value for the menu item is a path to an Axiom form.

Variable / value pairs are specified as follows:

```
Variable1=Value;Variable2=Value
```

Separate multiple variable/value pairs using semicolons. If a value contains a semicolon, then it must be preceded by a backslash (\) so that Axiom does not treat the semicolon as a delimiter. Equals signs within a value (such as to pass a filter criteria statement as a value) do not need to be specially treated.

When the user clicks on this menu item, the designated document variables are passed to the target form and can be returned in that form using GetDocumentInfo functions. For more information, see Passing document variables using a menu.

[HelpCode]

Optional. Specifies a help code to make available to the user when the item is the currently selected menu item. The code must be a valid form help code as defined in the Form Help Admin area of the Web Client. Only applies when the menu type is Tabular or Link.

If an item has a specified help code, then the help question mark icon displays in the far-right end of the menu. When a user clicks on the icon, a help panel opens along the right-hand side of the web page, displaying the help text for the designated help code. The help icon and behavior is the same as when using the Form Help component. This feature effectively embeds the Form Help component within the menu, so that it can automatically update to display help for the currently selected menu item.

If blank for a menu item, no help icon displays when that menu item is selected.

NOTES:

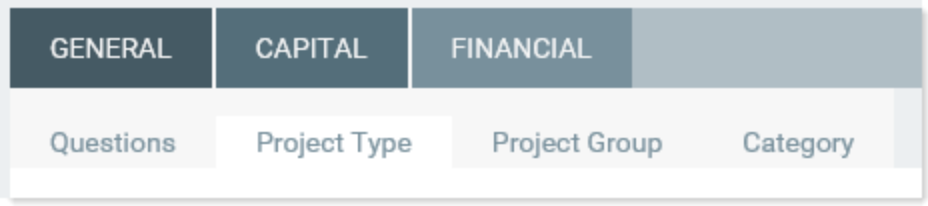
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example data source defines a menu with two levels.

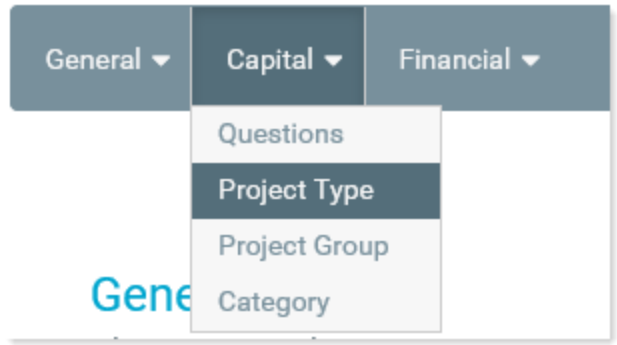
	A	B	C	D	E	F	G	H	I	J
2										
3		[Menu;Drivers]	[ID]	[ParentId]	[Name]	[Value]	[Tooltip]	[Disabled]	[Hidden]	[NewWindow]
4		[MenuItem]	General		General		General capital drivers			
5		[MenuItem]	Setup	General	Setup	\Axiom\File Groups\Capital Requests\[	General setup options			
6		[MenuItem]	Templates	General	Templates	\Axiom\File Groups\Capital Requests\[	Template options	TRUE		
7		[MenuItem]	Picklists	General	Picklists	\Axiom\File Groups\Capital Requests\[	Customize picklists	TRUE		
8		[MenuItem]	Capital		Capital		Capital-specific drivers			
9		[MenuItem]	Questions	Capital	Questions	\Axiom\File Groups\Capital Requests\[	Define capital project questions			
10		[MenuItem]	ProjType	Capital	Project Type	\Axiom\File Groups\Capital Requests\[	Define capital project types			

To use the Data Source Wizard to add the tags, right-click a cell and select **Create Axiom Form Data Source > Menu**. You can right-click a single empty cell to place the initial tags and then fill out the data, or you can have the data already in the spreadsheet and highlight the applicable data to add the tags. The cells in the row above the data and the column to the left of the data must be blank in order for Axiom to place the tags in sheet.

The resulting menu would display in the form as follows, depending on the configured type of the Menu component (Tabular or Drop-Down):

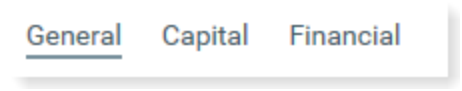


Tabular format



Drop-down format

The Link format does not support multiple-level menus, so the example data source shown above could not be used. However, if the data source only had top-level items with values, the menu would display as follows when using Link format:



Link format

Component properties

You can define the following properties for a Menu component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the component. You can select any defined Menu data source in the file. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Menu Type	The display format of the menu. Select one of the following: • **Drop-down** (default): The menu displays as a drop-down menu. If a top-level item has child items, the menu item expands downward to display the items. Subsequent levels expand to the right. • **Tabular**: The menu displays as a horizontal series of tabs. If a top-level item has child items, the child items display as a second horizontal series of sub-tabs underneath the top-level tabs. • **Link**: The menu displays as a horizontal series of hyperlinked labels. Link menus only support one level of menu items.
Selected ID	The currently selected item in the menu. This setting serves two purposes: • It specifies the initially selected item in the menu, when the user first opens the form. You can leave the setting blank to have no initial selection, or you can enter an ID as defined in the `[ID]` column of the data source. • When a user views the form and selects an item in the menu, the ID of the selected item will be submitted back to the source file and placed in this cell on the Form Control Sheet. Additionally, the corresponding value of the ID is written to the **Selected Value** cell. Other components can reference either cell in order to dynamically change the form based on the currently selected item in the menu. In certain cases the ID and value are not written back to the source file, such as when a menu item is used to open a URL in a new window. For more information, see Specifying the value for a menu item.

Item	Description
Selected Value	When a user selects an item from the menu, the defined value from the `[Value]` column of the data source is written to this cell on the Form Control Sheet (based on the selected ID). This makes it easier for other components, such as an Embedded Form component, to use the associated value for the selected ID. **NOTE:** This setting is only available on the Form Control Sheet. The Selected Value cannot be used to set a default value for the menu; the Selected ID should be used instead.
Parent Menu Buttons	Optional. The name of one or more Button components to display on the parent (top-level) menu bar. Only applies if the **Menu Type** is **Tabular**. Separate multiple names with semicolons. Buttons display in the order listed. This feature can be used to display a button "toolbar" on the right-hand side of the menu. For more information on how to set up this feature, see Displaying buttons on the menu.
Child Menu Buttons	Optional. The name of one or more Button components to display on the child (bottom-level) menu bar. Only applies if the **Menu Type** is **Tabular**. Separate multiple names with semicolons. Buttons display in the order listed. This feature can be used to display a button "toolbar" on the right-hand side of the menu. For more information on how to set up this feature, see Displaying buttons on the menu.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Menu components. Only the generic styles are available. Additionally, menu components do not have component-specific formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For

more information on these properties, see Position and size properties.

NOTE: The height of the component is fixed when using the tabular menu type. Changing the height property is not recommended and will not affect the display of the component (though it could potentially affect the menu hover behavior, if Axiom Software thinks that the component is taller than it actually renders). When using the drop-down menu type, the component will honor the height setting, though only the background color will expand to fill the space—the menu names will remain the same size.

Interactive behavior

The Menu component is intended to be used as in-form navigation, to change the contents shown in the form based on the currently selected item in the menu.

When a user clicks on an item in the menu, the ID for the item is submitted back to the source file, and written to the **Selected ID** cell on the Form Control Sheet. Additionally, the corresponding value for the item is written to the **Selected Value** cell on the Form Control Sheet. Both the ID and the value are obtained from the Menu data source.

NOTE: Menu components always auto-submit in response to clicking menu items, when the item is not being opened in a new window. If the item is opened in a new window, then no values are submitted and no form update occurs.

If you want the Axiom form to respond to the currently selected item, then you must set up the file so that something else in the file references the selected value and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Example

The Axiom form can contain an Embedded Form component to display a child form within the current parent form, and the values in the Menu data source can be file paths to Axiom forms. The Embedded Form component can be linked to the Menu component by using the **Source Menu Component** property of the Embedded Form component. This causes the Embedded Form component to automatically use the current selected value of the menu as its target form.

You could also use the selected ID or selected value of the Menu component in other ways, such as to define the text of a Label component. The **Text** field of the Label component could use a formula that references the **Selected ID** cell of the Menu component. As the user selects items in the menu, the text of the Label component would update to show the currently selected item.

Specifying the value for a menu item

When specifying the value for a menu item, use the following entries:

Value	Description
<Blank>	You can leave the value blank if the menu item is a parent item with child menu items. For more information on using parent and child menu items, see Creating multi-level menus. If the value is blank, clicking on the item expands the menu to show the child items. No other action occurs. The selected ID and value are left as is. If the menu type is drop-down, then it is required to leave the value blank for parent items.

Path to Axiom form	You can enter a file path to a form-enabled file, in order to open that form within an Embedded Form component, or to open it in a new window.
	
	The file path can be specified with or without the `document://` prefix. For example, either of the following paths are valid:
	
	`\Axiom\Reports Library\Forms\MyFile.xlsx`
	
	`document://\Axiom\Reports Library\Forms\MyFile.xlsx`
	
	The menu item behaves as follows:
	
	• If `[NewWindow]` is False, then the selected ID and value are written back to the source file. It is up to the form designer to configure an Embedded Form component to use the selected value.
	• If `[NewWindow]` is True, then the specified form is opened in a new window, and the current form is left as is.
	
	NOTES:
	
	• It is not possible to enter a path to a non-form-enabled file, in order to open a spreadsheet file in the Desktop Client. If you want to do this, you must use the GetDocumentHyperlink function to generate a URL to the spreadsheet file, and then use the URL as the value instead of a path.
	• If you want to open a non-embedded Axiom form in the current window (replacing the current form), you must use the GetFormDocumentURL function to generate a URL to the form, and then use the URL as the value instead of a path.

Value	Description
Component or layer name	You can enter the name of a component or a layer in the form, in order to dynamically show that component or layer when it is the selected value. Panel components and layers are most likely to be used here, since they can define entire "screens" worth of content. However, you could also use the menu to switch between showing certain charts, maps, or other components. When the menu item is selected, the selected ID and value are written back to the source file. The file must be configured to change in some way based on this value. For example, imagine that you have 3 Panel components that correspond to 3 menu items. When the selected value is Panel1, then Panel1 should be visible and the other panels should be hidden. It is up to the form designer to set up the necessary formulas to show and hide the panels. The `[NewWindow]` option does not apply when using component or layer names. All showing and hiding occurs within the current form. If set to True, an error occurs.
URL	You can enter a URL, in order to open that URL in the current window or a new window. The URL can be to a web site, or to an Axiom file (such as a URL created using GetFormDocumentURL). When the value is a URL, clicking on the menu item opens the target URL. No other action occurs. The selected ID and value are left as is. **NOTE:** This is the only way to open an Axiom spreadsheet file using the Menu component. If you generate a URL to the file using the GetDocumentHyperlink function, the link will launch the Desktop Client and open the spreadsheet file.

You can use other values as needed, as long as you set up the form to respond to the selected ID or value. For example, you might set the value to the menu ID or name, and then use formulas to show or hide rows in a grid based on the selected ID or value.

Creating multi-level menus

You can create multi-level menus by assigning a parent ID to a menu item. This causes the child item to display underneath the parent item on the menu. To do this, enter the ID of the parent item into the `[ParentID]` cell of the child item.

The following example shows a Menu data source using parent IDs to create multi-level menus. In row 4, the General menu item is a top-level parent item because it does not have an assigned parent ID. In row 5, the Setup menu item is assigned General as a parent ID, so it will display underneath the General menu item (as will the items in rows 6 and 7).

	A	B	C	D	E	F
2						
3		[Menu;Drivers]	[ID]	[ParentId]	[Name]	[Value]
4		[MenuItem]	General		General	
5		[MenuItem]	Setup	General	Setup	\Axiom\File Groups\Capital Requests\Drivers\Setup.xlsx
6		[MenuItem]	Templates	General	Templates	\Axiom\File Groups\Capital Requests\Drivers\Templates.xlsx
7		[MenuItem]	Picklists	General	Picklists	\Axiom\File Groups\Capital Requests\Drivers\Picklists.xlsx

See the following sections for multi-level design considerations by menu type.

Tabular

Tabular menus can only have two levels of menu items (parent and child). When a user clicks on a parent item, the child items display in a 2nd level of tabs, underneath the top-level items.

Parent items in tabular menus can optionally have an assigned value, or the value can be left blank. When a user clicks on the parent item in a tabular menu, the selected ID and value are handled as follows:

- If the parent item has an assigned value, that value is used.
- If the parent item does not have an assigned value, then the first child item is selected by default. If the user selects a child item in the sub-menu, that item will be remembered the next time the user clicks on the parent item in the current session.

Using the example data source shown previously, the first time the user clicks on General, the Setup child item is selected by default (because General does not have an assigned value). Now imagine the user clicks on Templates, and then later clicks on a different parent item. If the user goes back to General, now the Templates menu item is selected, because the user's previous selection in that sub-menu is remembered.

Drop-Down

Drop-down menus can have any number of levels. The behavior of the child items depends on the level of the parent:

- If the parent is a top-level menu item, the child items are shown in a drop-down menu that expands underneath the parent.
- If the parent is not a top-level menu item, the child items are shown in a side menu that expands to the right of the parent.

When using the drop-down menu, the user must first click on the parent item in order to view the child items. However, once a menu has been expanded, the user can hover their cursor over other menu items to view the child items.

Parent items in drop-down menus cannot have assigned values. If a parent item has an assigned value, then clicking on the parent item will activate the parent value instead of opening the drop-down menu.

Link

Link menus do not support multiple levels. Any items with parent IDs will not be accessible using the link menu.

Passing document variables using a menu

Using the `[DocumentVariables]` column in the data source, you can pass values to the target form. This feature only applies when the value for the menu item is a path to an Axiom form. The form can be opened as a child form within an Embedded Form component, or the form can be opened in a new window.

For example, you might want to use a menu to open the same child form repeatedly, applying different values each time the form is opened using a different menu item. Using the `[DocumentVariables]` column in the data source, you can set values for each menu item, and automatically apply those values when the user selects the menu item and opens the target form.

The following data source shows an example of passing variable values using a menu:

	A	B	C	D	E	F	G
1							
2		[Menu;MenuSource1]	[ID]	[ParentId]	[Name]	[Value]	[DocumentVariables]
3		[MenuItem]	1		Corporate	\Axiom\Reports Library\Dashboards\CorpDashboard.xlsx	
4		[MenuItem]	2		By Entity		
5		[MenuItem]	3	2	Entity 1	\Axiom\Reports Library\Dashboards\EntityDashboard.xlsx	Entity=1
6		[MenuItem]	4	2	Entity 2	\Axiom\Reports Library\Dashboards\EntityDashboard.xlsx	Entity=2
7		[MenuItem]	5	2	Entity 3	\Axiom\Reports Library\Dashboards\EntityDashboard.xlsx	Entity=3

In this example, IDs 3-5 all reference the same target form in the `[Value]` column. When a user clicks on ID 3 in the menu, the Entity value of 1 is passed to the target form. Then when a user clicks on ID 4 in the menu, the Entity value of 2 is now passed to the target form. Assuming that the target form is set up to filter data queries based on the value of Entity, the form will show different data depending on the currently selected ID in the menu.

The variable values can be returned in the target form by using the GetDocumentInfo function. For example:

```
=GetDocumentInfo("Variable","Entity")
```

This returns the passed value of Entity when the target form is opened.

Displaying buttons on the menu

You can display one or more Button components on the right-hand side of a tabular menu, on either the top-level parent menu or the bottom-level child menu. This feature can be used to display a button "toolbar" on the menu.

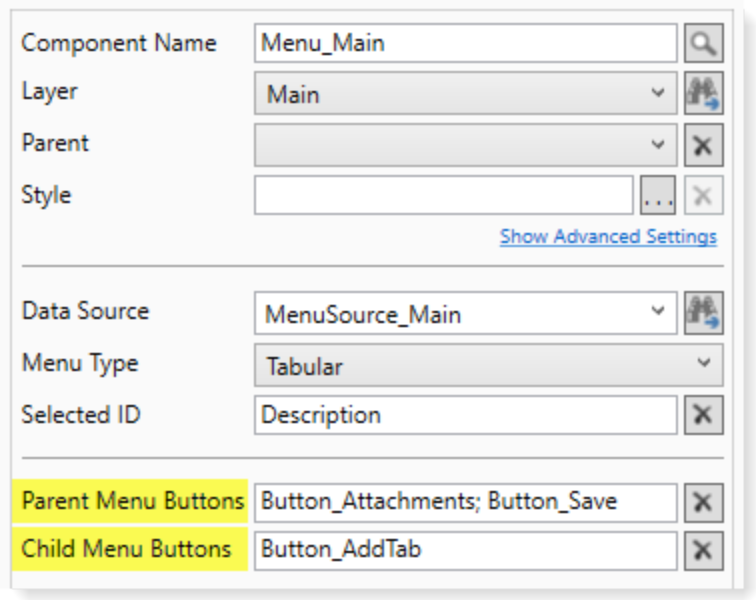
To display buttons on the menu, do the following:

- Create a dedicated layer to hold the buttons, and configure that layer to not be visible on the form. On the Form Control Sheet, set **Visible** to **Off**.
- Create the Button components, and place them on the layer. Although the layer is configured as not visible, the buttons on the layer must have **Visible** set to **On** in order to show on the menu.

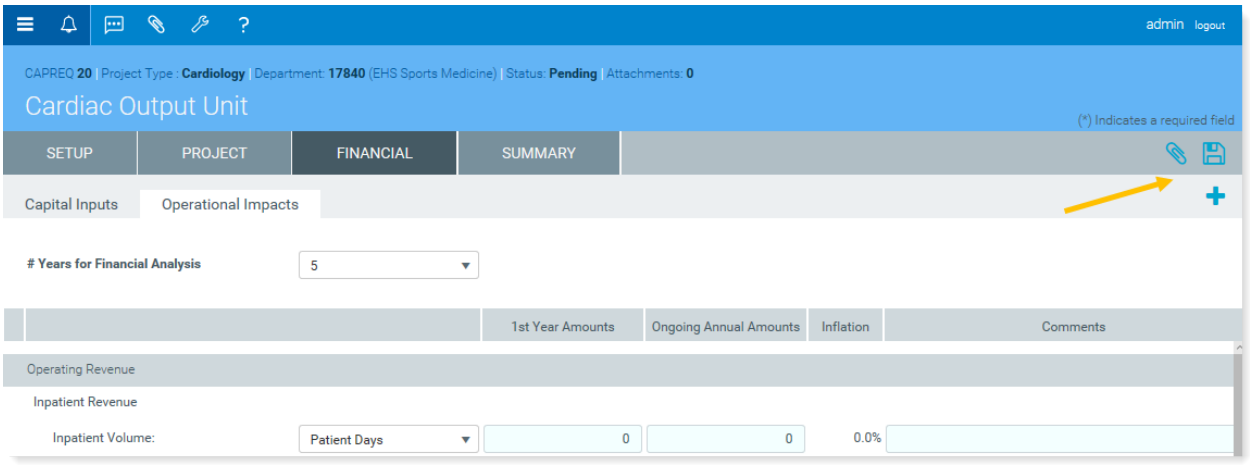
  You can use any type of button, but link-style buttons often look best in this environment. You can use text, symbols, and images, but remember that the buttons must fit in the space allotted to the menu bar.

  The position of the button on the layer does not matter, because the button will be automatically positioned on the right-hand side of the menu bar. However, the size of the button matters. The button height should be no greater than 40px (the height of the menu bar). The button width should be no wider than necessary to hold the button text and/or symbol, because extra blank space may throw off the spacing of the buttons.
- Enter the Button component names into either the **Parent Menu Buttons** property or the **Child Menu Buttons** property of the Menu component, depending on where you want the buttons to display. Separate multiple button names with semicolons.

The following example shows a Menu component that is configured to display buttons on the menu. Two buttons will be shown on the top parent menu bar, and one button will be shown on the bottom child menu bar.



When the menu is rendered, the buttons are displayed on the far right-hand side, as shown in the following screenshot. Because the buttons are integrated with the Menu component, the buttons are automatically positioned on the menu.



In this example, the buttons are all link-style buttons using symbols instead of text. Alternatively the buttons could have used text such as "Attachments" and "Save", with or without the symbols. Remember that the Tooltip property on the Button components can be used to display helper text when a user hovers over a button.

If the `[HelpCode]` property is populated in the Menu data source, this causes a question mark icon (help button) to display in the far right of the menu bar. In this case, the parent and child menu buttons are moved slightly to the left, with extra space between the "toolbar" buttons and the help button.

Although you can instead simply place a Button component on top of the Menu component on the form canvas, this approach is not ideal. It can be difficult to manually position the button properly, and since the button is not actually part of the menu, the button will not automatically adjust to the menu (such as when the menu tabs overflow). It is recommended to use the official method of placing buttons on the menu, so that the menu and buttons adjust together.

Menu display behavior notes

- When using the tabular or link menu, if the number of menu items on a level exceeds the available horizontal space for the menu, the remaining items are displayed in a drop-down list. This list is accessible by clicking the **...** (ellipses) that display at the far right end of the menu.
- When using the drop-down menu, the menu overflows to additional rows if all items cannot be displayed on the first row.
- When using the tabular or link menu, the currently active item is indicated on the menu. For tabular menus, the active top-level item uses a dark background, and the active child item uses a light background. For link menus, the active item is displayed with an underline.
- The link menu is shown in dark gray font with gray underline. Currently, there is no way to style the link menu to display as blue hyperlinks.

Process Summary component

The Process Summary component can be used to display information about users' current process tasks. The component is automatically filtered to show the tasks for the current user, for a specified plan file process. The goal is to direct the user's attention to their active tasks, and allow the user to easily take action on those tasks.

This component is primarily intended to be used in form-enabled Home files for the Web Client, to provide users with a way to easily see and manage their process tasks in the Web Client. The component could also be used in a form-enabled Home file for the Desktop Client. However, in this case you may want to disable the Process task pane (or only show it to certain users), so that end users are not confused by having two user interfaces for viewing and completing process tasks.

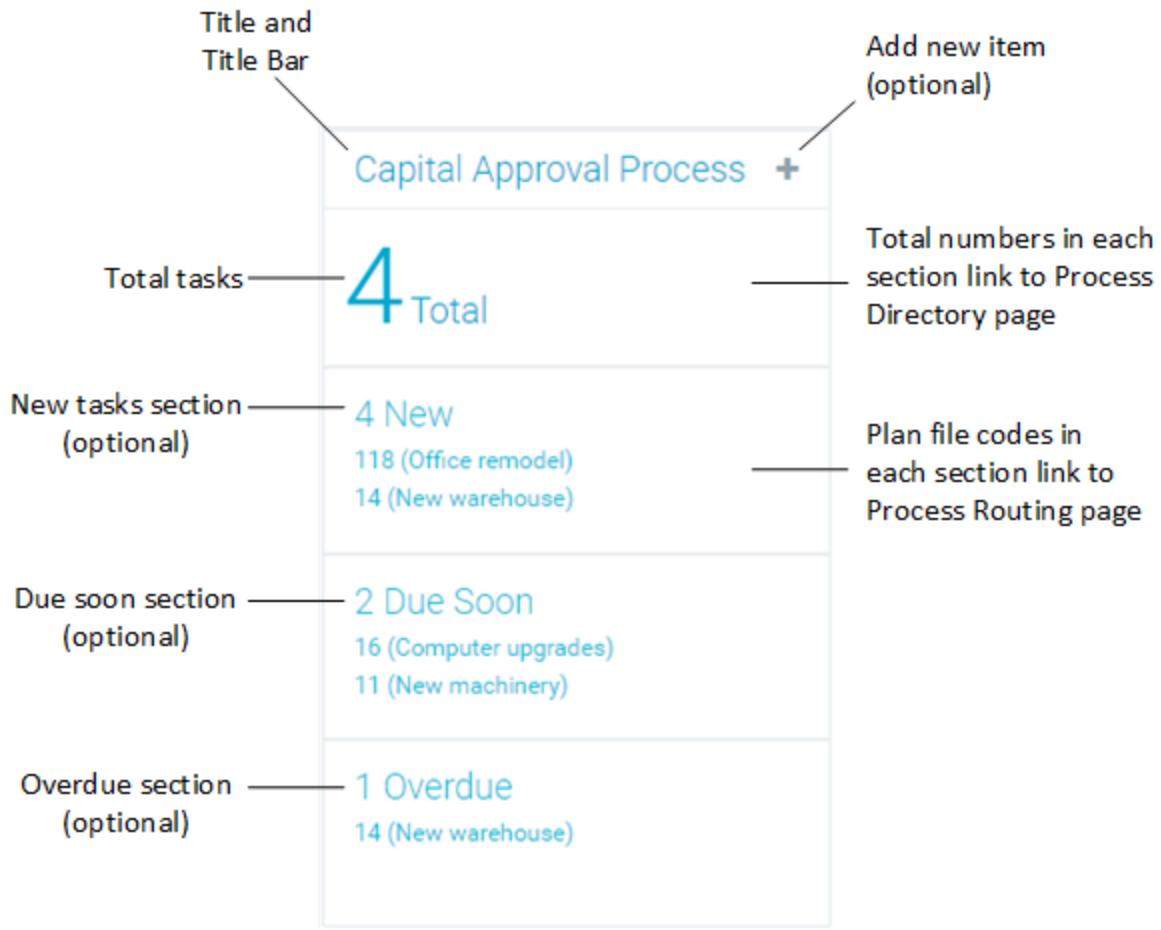
Component overview

The component displays the following information for the current user, for a specified plan file process:

- The total number of tasks for the user
- The number of new tasks for the user (optional)
- The number of tasks due soon for the user (optional)
- The number of overdue tasks for the user (optional)

For each of the optional sections, the component displays the plan codes and descriptions for the plan file tasks in that category (up to a maximum number of items). You can also optionally allow users to create new plan files for on-demand file groups from this component.

The following screenshot shows an example component with the major sections and features annotated:



NOTE: The built-in hyperlinks to the process web pages always open in the Web Client (the user's browser), even if the form is open as an embedded web tab in the Desktop Client.

Component properties

You can define the following properties for a Process Summary component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Selected Process	Specifies the process to display in this component, based on a selected file group. Click the **Choose a plan file process** button **[...]** to select a file group or a file group alias. The selected file group must have a designated **Plan File Process** in its file group properties. The Process Summary component will use that process. If the designated process for the file group changes, the component will automatically update to use the new process. If the selected file group is a file group alias, then the component will use the process for the file group that the alias currently points to. If the alias is changed to point to a different file group, the component will automatically update to use the new file group and its designated process. In the Form Designer and Form Assistant, the Selected Process field displays the name of the plan file process that it is using, not the name of the selected file group or alias. However, if you look at the corresponding Form Control Sheet entry, the file group is indicated using shortcut syntax such as `filegroup://Capital Requests?_tid=110`.
Title Text	The title text for the component. This text displays in the title bar for the component within the Axiom form, if the title bar is enabled. If the title bar is disabled, then this text does not display at all in the form.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.

Item	Description
Collapse Height	Specifies whether the component automatically collapses in height if the configured component height is greater than the content to be shown in the component. You should leave this option disabled if you want the component height to always be the same, no matter how much content is available to display. If the content does not fill the component, then there will be blank space between the content and the bottom edge of the component. If you enable this option, then when the content does not fill the component, the component will auto-shrink to fit the content instead of maintaining its configured height. The component behavior when the content exceeds the available height is always the same: excess content will be excluded. In practice, this means that the number of plan file tasks in each section will be limited to whatever amount will fit in each section, rather than honoring the maximum tasks per section.
Orientation	Specifies how the sections in the component are presented within the form: • **Vertical** (default): Sections are stacked vertically. The component is tall and thin. • **Horizontal**: Sections are presented side-by-side horizontally. The component is short and wide. If you do not include any of the optional sections, then the component is essentially a square and the orientation does not matter.
Show new item button	Specifies whether users can create new on-demand plan files from the title bar of the component. By default, this is disabled. If enabled, then a plus icon displays in the title bar. Users can click the plus icon to create a new on-demand plan file in the file group. The file group is determined based on the specified process for the component. The plus icon only displays if: • The file group is an on-demand file group. • The file group uses an **Add File Form** to create new plan files. • The current user has the **Create New Records** permission for the file group. • The title bar is enabled for the component. When a user clicks the plus button, the designated Add File Form opens. The user can use this form as normal to create a new plan file.

Item	Description
Task count text	Defines text for the total tasks section. This section always displays the total number of current tasks for the current user, for the specified process. This section is always included. By default, this is set to `Total`. So if a user has 5 current tasks, this section displays the text **5 Total**. You can customize the text by entering different text here, but you cannot omit the total count number. Also, the text cannot be omitted entirely—if you make this property blank, the component will display using the text **Total** by default.
Max tasks per section	Specifies the maximum number of plan file tasks to show per section. By default, this is set to 2. If a section has active tasks, then the plan file names for those tasks display in the section, up to the maximum number of tasks per section. The plan file tasks display as hyperlinks that can be used to open the Process Routing page for that plan file. For more information, see Viewing process status using process web pages. The number of tasks that can be displayed per section depends on the overall height of the component (for either orientation), and on the number of optional sections that are configured to display (when using vertical orientation). The space available for each section is divided equally among the optional sections. If you are using all three sections, each section gets one third of the available height. If the maximum tasks do not fit within the available height, then a lesser number of tasks will display.
Show new tasks	Specifies whether to include a section for new tasks. This option is selected by default, which means that details about new tasks display in the component. If you disable this option, then the new tasks section does not display in the component. The number of days that a task is considered new is configured within the process definition. By default, tasks are considered new for 2 days once they are started.
New tasks header	Defines header text for the new tasks section. This setting only applies if **Show new tasks** is enabled. By default, this is set to `{count} New`. For example, if the user has two new tasks, the header will display as **2 New**. If the user has no new tasks, then the header will display as **0 New**. You can edit this text as desired, but if you want the number of new tasks to display, you must use the variable `{count}`.

Item	Description
Show tasks due soon	Specifies whether to include a section for tasks that are due soon. This option is selected by default, which means that details about tasks that are almost due display in the component. If you disable this option, then the due soon section does not display in the component. The number of days that a task is considered due soon is configured within the process definition. By default, tasks are considered due soon for 2 days until their due date.
Due soon header	Defines header text for the due soon section. This setting only applies if **Show tasks due soon** is enabled. By default, this is set to `{count} Due Soon`. For example, if the user has two tasks that will be due soon, the header will display as **2 Due Soon**. If the user has no tasks that will be due soon, then the header will display as **0 Due Soon**. You can edit this text as desired, but if you want the number of new tasks to display, you must use the variable `{count}`.
Show overdue tasks	Specifies whether to include a section for overdue tasks. This option is selected by default, which means that details about overdue tasks display in the component. If you disable this option, then the overdue tasks section does not display in the component.
Overdue tasks header	Defines header text for the overdue tasks section. This setting only applies if **Show overdue tasks** is enabled. By default, this is set to `{count} Overdue`. For example, if the user has two tasks that are past their due date, the header will display as **2 Over**. If the user has no tasks that are overdue, then the header will display as **0 Overdue**.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Process Summary components. Only the generic styles are available. Most styling for the component is controlled by the form-level skin.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Titled Panel template

The Titled Panel is intended to be used as a template to create forms with consistent title and content areas. Technically it is not a unique component type; rather it is a collection of preconfigured components. However, you add it to a form in the same way you add other components.

The Titled Panel "template" consists of the following components. All of these components will be added to the form when you drag and drop a Titled Panel on to the form canvas:

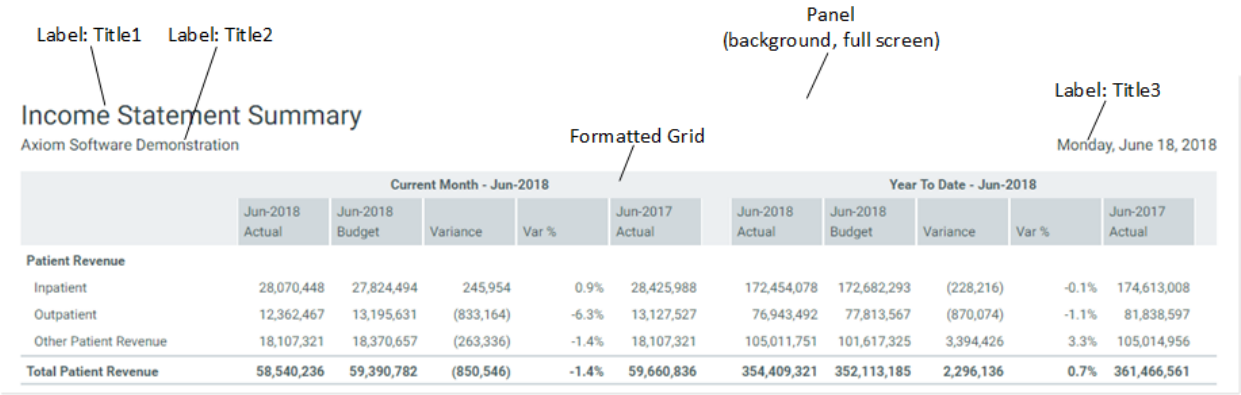
- A parent Panel component that is used to position and group the other components.
- Three Label components at the top of the panel, which are used to display a main title and two subtitles.
- A Formatted Grid component, which is intended to display the form contents.

By default, the Titled Panel template is intended to use the entire page. When you drag and drop the Titled Panel onto the canvas, the parent Panel component is automatically configured to fill the canvas. It is recommended to start with a new form file when using this template, as the panel will cover up any existing components on the canvas. If you need to add other components to the form, it is best to add them after the panel is already in place, so that the components are automatically assigned to the panel as you place them on the canvas.

The Titled Panel component can be used for a variety of purposes. For example, you can use the Formatted Grid component to display reporting data or to gather data inputs. Or, you can delete the grid and instead create your own content area. For example, you might populate the area with various charts to create a dashboard-style report, or create a data input area using individual form controls.

▶ Titled Panel example

The following screenshot shows an example of a form that was created using the Titled Panel. The grid is configured to show reporting data.



	Current Month - Jun-2018					Year To Date - Jun-2018				
	Jun-2018 Actual	Jun-2018 Budget	Variance	Var %	Jun-2017 Actual	Jun-2018 Actual	Jun-2018 Budget	Variance	Var %	Jun-2017 Actual
Patient Revenue										
Inpatient	28,070,448	27,824,494	245,954	0.9%	28,425,988	172,454,078	172,682,293	(228,216)	-0.1%	174,613,008
Outpatient	12,362,467	13,195,631	(833,164)	-6.3%	13,127,527	76,943,492	77,813,567	(870,074)	-1.1%	81,838,597
Other Patient Revenue	18,107,321	18,370,657	(263,336)	-1.4%	18,107,321	105,011,751	101,617,325	3,394,426	3.3%	105,014,956
Total Patient Revenue	**58,540,236**	**59,390,782**	**(850,546)**	**-1.4%**	**59,660,836**	**354,409,321**	**352,113,185**	**2,296,136**	**0.7%**	**361,466,561**

Component properties

When you drag a Titled Panel on the Axiom form canvas, the following preconfigured components are added to the form. See the previous section for an example of how these components are positioned and formatted.

By default, styles are used to set the size and position of all components, so the component size and position properties are blank. The components are also locked on the canvas, so that you do not accidentally move or resize the components as you are working in the Form Designer. If you want to move or resize a component, you can do so by manually adjusting the size and position properties in the advanced component settings. Alternatively, you can disable **Lock Layout** so that you can edit the size and position on the canvas.

For more information about any of the styles used by the components, select the component in the Form Designer and then open the **Choose Style** dialog to view the effective formatting of the style. If necessary, you can use advanced settings to override certain style properties—for example, if you want the panel background to be a different color.

You will not see any entry in the Form Control Sheet for Titled Panel. This does not exist as a separate component; it only exists in the component sidebar as a way to place these preconfigured components on the canvas.

Panel component

The Panel component uses the following styles:

- **docked-to-container**: This style sets background formatting and causes the panel to dynamically fill the entire page.
- **page-padding**: This style adds standard page-level padding along all sides of the page, so that content does not extend all the way to the edges.

The panel provides an easy way to manage the positioning of all the other components. If you intend to use the default behavior of a full-page panel, then there should be no need to adjust the positioning of the panel.

If you want to use the panel as a partial-page component, then you must manually adjust the component's position and/or size properties as appropriate. These component-level settings will override the style-level settings. For example, you could set the **X Position** to 200px to move the entire panel over to the right, so that you could display some other content as a left-hand sidebar. Because the other components in the panel are positioned and sized relative to the panel, they will automatically adjust to the new settings.

Label components

Three different Label components are used to create the main title and the subtitles:

- **Title1**: This defines the main title at the top of the page. It uses the style **page-title**. This style applies title formatting and sizes the title to fill the full width of the page.
- **Title2** and **Title3**: These define the two sub-titles located underneath the main title. Both use the style **page-subtitle** to define sub-title formatting and vertical positioning. Title2 is left-aligned and Title3 is right-aligned. Title3 also uses **UpperRight** as its reference location so that it is positioned relative to the right side of the parent panel.

If you want a second row of subtitles, you can duplicate the existing subtitles and then manually modify the **Y Position** to move them down. You would have to also modify the Y Position of the Grid to push it down as well.

Formatted Grid component

By default, the Formatted Grid component is assumed to provide the content of the form. It could be used to display reporting data or gather user inputs. You must create the data source for the grid as normal and assign it to the component.

The grid uses the style **titledpanel-body**. This style positions the component vertically and horizontally, and causes it to extend to fill the remaining width and height of the parent panel. If you add or remove titles, or modify the sizing of the titles, you will need to manually override the **Y Position** to move the grid up or down as appropriate.

Using different components for the content

If you do not want to use a Formatted Grid component to provide the form contents, you can delete the grid and populate the page as desired with other components. If you do this, it is recommended to first replace the grid with a Panel component. This makes it easier to control the positioning and sizing of the child components and ensure that they remain in the designated "content area" outlined by the panel.

To do this:

- Drag and drop a Panel component onto the canvas (after deleting the grid).
- Set the **Style** for the panel to **titledpanel-body**. This gives it the same style as the original grid.
- Click **Show Advanced Settings** and clear out **X Position**, **Y Position**, **Width**, and **Height**. This allows the component to inherit size and position from the style. You may also want to enable **Lock Layout** so that you do not accidentally move or resize the panel later.

- Refresh the canvas and now you should see the panel take up the same space that the grid used to.
- In most cases, you should also disable **Show Title Bar**, so that the panel is just a design tool for organizing components instead of visible on the form.

You can now drag and drop other components into this panel, and the size and position of those components can be controlled relative to the panel. For example, you can place two charts side by side, and set the **Reference Location** of one to **UpperLeft** and the other to **UpperRight**. Set the **X Position** and **Y Position** for both charts to 0, so that they align with the top of the panel and their respective reference locations. Lastly, set the **Width** and **Height** of each to 50%. Now you have two charts of equal width side by side, filling up the top half of the panel. This is just an example of how you can control size and position of child components within the panel.

Wizard Panel component

The Wizard Panel component is an interactive component that displays a series of ordered screens to the form user. Each screen is a step in the wizard. The steps are intended to be used to gather various user inputs and then apply those inputs toward a particular action, such as creating a new plan file, or saving data to the database, or triggering a Scheduler job for execution.

The Wizard Panel is intended to streamline the process of creating guided forms. It provides a standardized look and feel so that you do not have to design the wizard "container." It also provides a built-in methodology for moving through steps and showing the appropriate content, so that you do not have to manually set up the step contents to dynamically show and hide.

Defining a wizard panel requires several steps. The basic setup requires the following:

- Creation of a WizardPanel data source in the spreadsheet to define the list of steps to show in the wizard and their basic properties.
- Placement and configuration of a Wizard Panel component on the Axiom form canvas.

Additionally, you must define the contents to display for each step. There are two different approaches to define step contents. For more information, see Defining the wizard contents for each step.

By default, the Wizard Panel component is intended to use the entire page. When you drag and drop the component onto the canvas, it is automatically configured to fill the canvas. It is recommended to start with a new form file when using this component, as it will cover up any existing components on the canvas. In many cases you will not need to add any more components to the canvas, assuming that you plan to use the thematic Formatted Grid that is also placed on the canvas by default. But if you do want to use other components, it is best to add them after the Wizard Panel is already in place, so that the components are automatically assigned to the panel as you place them on the canvas.

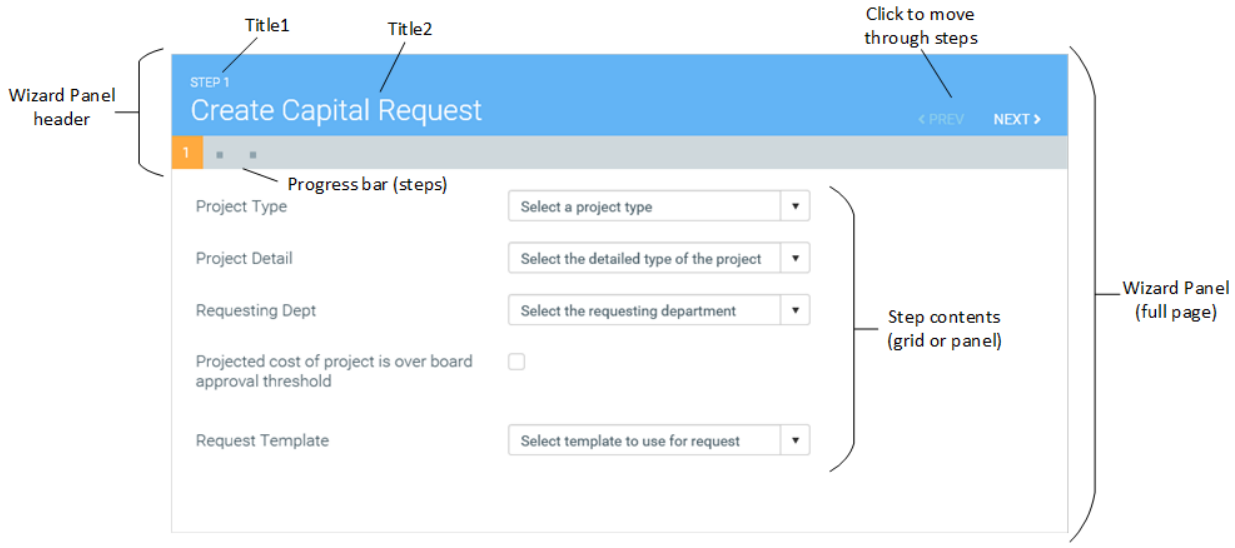
NOTES:

- If your form uses a legacy skin (any skin other than Axiom2018), then the theme for the form must be set to **Wizard** in order for the Wizard Panel to display property. This theme assignment occurs automatically when the Wizard Panel is first added to the form, and should not be changed. This does not apply to the Axiom2018 skin, where the necessary Wizard Panel formatting is automatically applied to forms that contain a Wizard Panel.
- The Wizard Panel component is a Panel component with special default settings and support for wizard-related features. The panel aspect of the component behaves just like a normal panel, as far as placing other components on the panel and becoming the parent for those child components. For more information on this behavior, see Using panels to group and position components.

Wizard Panel overview

The Wizard Panel component uses pre-set formatting and supports various built-in features. These features leverage the settings defined in the WizardPanel data source and the component properties to present a full-featured and polished "wizard" to the user.

The following screenshot illustrates the Wizard Panel's formatting and features:



The steps that drive the wizard are defined in the WizardPanel data source. Within the data source, you define step properties such as:

- The value to use for each step (to show the step contents)
- The step titles (Title1 and Title2)
- The step tooltip text for the progress bar
- Whether the step is required to be completed before the user can move to the next step (and if

so, whether the step is currently complete)

- Whether the step is hidden (for wizards with variable steps)

As users move through the steps using the **Prev** and **Next** buttons in the header (or by clicking the step icons in the progress bar), the titles for the current step display at the top of the page, and the progress bar updates to show the current step. Additionally, the value for the step is written to the Selected Value of the Wizard Panel component.

To display the wizard contents, this example uses the thematic Formatted Grid component that is present in the Wizard Panel by default. When using this approach, the step values are all names of Grid data sources. The Formatted Grid component is configured to read its data source from the Selected Value of the Wizard Panel. Therefore as the Selected Value changes, so does the content of the grid, by pointing to a different data source.

Alternatively the step values can be Panel component names. When using this approach, only the panel that corresponds to the current Selected Value is shown in the wizard. For more information, see Defining the wizard contents for each step.

Data source tags

A Wizard Panel component must have a defined data source in the file to define the list of wizard screens and their basic properties. The tags for the data source are as follows:

Primary tag

`[WizardPanel;`*`DataSourceName`*`]`

The DataSourceName identifies this data source so that it can be assigned to a Wizard Panel component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Step]

Each row flagged with this tag defines a screen to display in the wizard. Steps are displayed in the wizard in the order they are found in the data source, from top to bottom. Users navigate through the wizard steps by clicking the built-in Prev and Next buttons, or by clicking the step icons in the progress bar.

If a step is hidden using `[HideStep]`, the wizard will skip that step in the step progression, and the hidden step will not be reflected in the progress bar. The total number of steps is re-evaluated at each form update, so that steps can dynamically show and hide.

Column tags

[Value]

This column contains the value used to define the contents of this step. As the user moves through the wizard steps, the value for the current step is written to the **Selected Value** property of the Wizard Panel component.

This column should contain either Formatted Grid data source names or Panel component names, depending on the approach you are using to define the panel content. For more information, see Defining the wizard contents for each step.

[Title1]

This column defines the first title for each step. This title displays in small text directly over the second title. See the screenshot in the previous section for an example. This title is not strictly required, but the wizard header is designed to show two titles.

This title is also used as the step tooltip if no text is defined in the `[StepTooltip]` column.

[Title2]

This column defines the second title for each step. This title displays in large text and is the most prominent title in the header. See the screenshot in the previous section for an example. This title is not strictly required, but the wizard header is designed to show two titles.

[StepTooltip]

Optional. This column defines a tooltip to display when the user hovers over the corresponding step icon in the status bar. The icon is a number for the current step, and a dot for all other steps. If this column is omitted or blank, the `[Title1]` text for the step displays instead.

[IsRequiredToMoveNext]

Optional. This column specifies whether this step must be completed before users can move on to the next step (True/False).

- If omitted or False, then users can move to the next step at any time. They are not required to complete anything on the current step before they can move.
- If True, then users must complete one or more items in the current step before they can move to the next step. In this case, the Next button is disabled in the wizard header until the `[IsComplete]` column for the current step is True.

[IsComplete]

Optional. This column determines whether this step is complete (True/False). It is used in conjunction with the `[IsRequiredToMoveNext]` column to determine whether the wizard can progress to the next step.

If the step is required, then the `[IsComplete]` column should contain a formula that determines whether the necessary inputs for this step have been completed. The logic that determines whether the step is complete is up to you.

In order for this formula to resolve based on user inputs, the input controls used for this step must be set to auto-submit, or the step contents must include a Button component (or a Button tag in a Formatted Grid component) to submit the step values back to the source file.

[HideStep]

Optional. This column specifies whether this step is hidden in the wizard (True/False). If omitted, the step is visible.

This column can be used to dynamically show and hide steps based on user inputs in other steps. For example, if the wizard is used to create a capital request, you may require additional steps if the request is projected to cost over a certain dollar amount. You can use a formula in this column to dynamically show or hide the additional steps as needed.

When the form is initially opened, the progress bar in the header shows the number of all currently visible steps. If a step is hidden, it will not be reflected in the status bar, and it will be skipped in the step progression. Whenever the form is updated, the currently visible steps will be re-evaluated and the progress bar and step progression will update accordingly.

If the steps in the wizard may change dynamically, then you should disable **Allow Step-Specific Navigation** in the component properties. The step-specific navigation in the progress bar will not work as expected if the number of steps change after the user has selected a specific step to navigate to.

NOTE: Make sure to work through the logical progression of steps when using hidden steps as well as required steps. It is possible to set up a configuration where a previously hidden step becomes visible, and that step is required, but that step is "earlier" than the user's current location in the step progression. In this case there is no way for the wizard to indicate that the user needs to go back and complete the prior step.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows sample wizard steps flagged in a report.

	A	B	C	D	E	F	G	H	I
3									
4		[WizardPanel; MyWizard]	[Value]	[StepTooltip]	[Title1]	[Title2]	[IsRequiredToMoveNext]	[HideStep]	[IsComplete]
5		[step]	Step1!Step1		Step 1	Create Capital Request	TRUE		FALSE
6		[step]	Step2!Step2		Step 2	Create Capital Request	TRUE		FALSE
7		[step]	Step3!Step3		Step 3	Create Capital Request	TRUE		FALSE

To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > Wizard Panel**. You can also highlight a range of cells first and then use the wizard. Axiom Software will add the tags as displayed in the example above. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The resulting wizard for this example data source would appear as shown in the previous section.

Component properties

You can define the following properties for a Wizard Panel component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the component, to define the steps in the wizard and their properties. A data source must be tagged within the file as detailed in the previous section, and then selected for the component. You can select any WizardPanel data source defined in the file. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Selected Value	The currently selected value for the Wizard Panel component. This setting is used to associate each step in the wizard with the content to display in the panel. As the user clicks through the wizard steps, the corresponding value from the `[Value]` column in the data source is written to this field. If the Selected Value is left blank, the value for the first step in the WizardPanel data source is assumed as the selected value when a user opens the form. If you want the wizard to start at a step other than the first step, you must enter the value for the desired step. The values should be either Grid data source names (if using the default Formatted Grid component) or Panel component names. As the user moves through the steps of the wizard, the contents will update to show the current Grid data source or panel. For more information, see Defining the wizard contents for each step.

Item	Description
Update Selected Values	Specifies when the selected value for the Wizard Panel component is updated after a user navigates to a different step. Select one of the following: • **Form - After AQ Refresh** (default): The selected value for the wizard is set after Axiom queries are refreshed in the form. This option is useful if you want to use the current state of data to dynamically determine whether a step is complete or whether it should be active. **NOTE:** When using this option, you cannot dynamically enable or disable Axiom queries based on the current step, because the current step is determined after Axiom queries are already run. • **Form - After Updating Values**: The selected value for the wizard is set after updated values are submitted to the form (but before Axiom query data is refreshed). This option is useful if you want to enable or disable Axiom queries based on the current step. These options correspond to the processing steps used by command adapters to determine when the command is run during the form update process. Behind the scenes, the Wizard Panel uses an internal command adapter to set its current step.
Allow Step-Specific Navigation	Specifies whether users can use the progress bar at the top of the wizard to navigate to specific steps. By default, this option is enabled, which means that users can click on the dots in the progress bar to navigate directly to the corresponding step in the wizard. If this option is disabled, then the progress bar is for display only. It shows users where they are in the current step progression, but they cannot use the progress bar to navigate to specific steps. Users must use the Back and Next buttons to navigate. You should disable this option if you are using the `[HideStep]` feature in the data source to dynamically enable or disable certain steps. The navigation in the progress bar will not work as expected if users attempt to navigate to specific steps but the number of steps is dynamic.
Title Text	The title text for the component. This text displays in the header bar for the component within the Axiom form, if the title bar is enabled. By default the title bar is disabled for Wizard Panel components, so this text does not display at all in the form.

Item	Description
Show Title Bar	Specifies whether the title bar is visible. By default this is disabled, which means that the title bar and the title text do not display on the wizard. If enabled, then the title bar and title text will display on the wizard. When using the default configuration of the Wizard Panel component, it is recommended to leave this option disabled. By default the wizard fills the entire form window, so the title bar is typically unnecessary and the title border is not visible.
Overflow	Specifies the behavior if child components extend beyond the visible area of the panel. Select one of the following: • **Visible** (default): Child components are visible beyond the panel edges. • **Hidden**: Any part of a child component that extends beyond the panel edges will be hidden. This may cause child components to appear "cut off." • **Scroll**: Scroll bars are added to the panel so that the entirety of any child components can be viewed by scrolling within the panel.

NOTE: The **Child Layout** property for Panel components does not apply to the Wizard Panel.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

To define the component formatting, you can assign one or more styles to the component. Styles can impact formatting properties such as fonts, borders, and colors.

If you do not want to apply a style to this component, or if you want to override one or more formatting properties in an assigned style, click the **Show Advanced Settings** link underneath the **Style** box to display the individual formatting properties. For more information on defining individual formatting properties for a component, see Formatting overrides for Axiom form components.

Item	Description
Style	By default, Wizard Panel components are automatically assigned the **docked-to-container** style. This style is used to set the size and position of the panel, as well as several formatting properties. In the majority of cases, you should leave the assigned style as is. Although you can remove or change the style, it is not recommended to do so. The Wizard Panel component is designed to be used with this assigned style. The colors used by the Wizard Panel cannot be changed by use of styles.
Theme Override	(Deprecated.) The theme to use for the component instead of the form-level theme. If left blank, the component uses the form-level theme. This setting should be left blank unless you need to override the form theme. Generally speaking, themes should be set at the form level and only overridden at the component level when necessary. This setting is available in the advanced component properties (click **Show Advanced Settings** under the **Style** box). On the Form Control Sheet, the setting displays using the name **Theme Override**. **NOTE:** This setting only applies if your form uses a legacy skin (any skin except the default Axiom2018). The Axiom2018 skin does not use themes.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

When you drag and drop a Wizard Panel component onto the canvas, the position and size properties are automatically left blank so that the panel can inherit these properties from the **docked-to-container** style. This style sets the x- and y-position of the component to 0px so that it is placed in the top left corner, and sets the width and height to **dock** so that the panel fills the full page. The panel is also locked by default (**Lock Layout** enabled) so that you cannot accidentally move it when working on the form canvas.

In the majority of cases, you should leave the position and size properties as is. Although you can override these settings—for example, to move the entire panel to the right so that you can have space for a left-hand sidebar—it is not recommended to do so. The Wizard Panel component is designed to be used as a full-page screen.

Interactive behavior

The Wizard Panel includes interactive elements that can be used to change the current contents shown in the wizard.

- The **Prev** and **Next** buttons can be used to move to the previous or next steps in the step progression.
- The progress bar can be used to navigate to specific steps, by clicking the step dots.

When a user clicks one of these buttons, the Wizard Panel always triggers a full form update (in other words, the wizard always uses auto-submit behavior). The **Selected Value** field for the Wizard Panel becomes populated with the value that corresponds to the currently selected step, as defined in the WizardPanel data source. For example:

- If the user is currently on the second step and clicks Next, the Selected Value field is populated with the value for the third step.
- If the user clicks the third button in the progress bar, the Selected Value field is populated with the value for the third step.

When the form update is complete, the wizard will show the contents for the third step.

The "third step" is determined dynamically each time the form update is triggered, based on the currently visible steps in the data source. This is determined by the `[HideStep]` column. If no steps are hidden, then the third step is the third row of the data source. But if some steps in the data source are hidden, then the third step might be the fourth row of the data source (for example if the third row is currently hidden).

NOTE: By default, the selected value for the Wizard Panel is not set until after Axiom queries are run. This means that you can use the returned data to determine whether a step should be shown or hidden via the `[HideStep]` column. However, this also means that you cannot dynamically enable or disable an Axiom query based on the current step, because the queries are already run by the time the current step is determined. If desired, you can change the timing of when the selected value is set, using the **Update Selected Values** property on the component.

In most cases the contents of each wizard step are also interactive, but that interactivity is governed by the specific components used to define the contents. For example, if you are using a Formatted Grid component to define the step contents, you must decide if you want auto-submit to be enabled for the grid or not. If auto-submit is not enabled for the step contents, then you must decide whether the contents can be submitted when the user clicks a Wizard Panel button, or if you need a separate button in the step contents to trigger the submit. If the step contents are required (using the `[IsRequiredToMoveNext]` column), then you must submit the step contents separately so that the step can resolve to completed in the data source (using the `[IsComplete]` column) and therefore enable the Next button in the wizard header.

Defining the wizard contents for each step

There are two different approaches you can use to define the contents for each step of the wizard:

- **Thematic grid**: You can use a single thematic Formatted Grid component for the contents, and then dynamically change the data source used to populate the grid for each step. Thematic grids support a variety of input controls such as combo boxes, check boxes, text boxes, and buttons, so it is possible to create robust screens with just this single component type. The default Wizard Panel component includes a preconfigured thematic grid that can be used for this purpose. For more information, see Using a formatted grid for Wizard Panel content.
- **Panels**: You can use a series of regular Panel components with child components to create the contents, and then dynamically show the appropriate panel for each step. The main advantage of this approach is that you can use any component in your wizard screens (including thematic grids). For more information, see Using panels for Wizard Panel content.

Your choice determines how the `[Value]` column should be populated in the WizardPanel data source. If using a grid, the values should be Grid data source names, in the format: *`SheetName`*`!`*`DataSourceName`*. If using panels, the values should be Panel component names.

Completing the wizard

After the user has completed all necessary inputs, you may need an action to "complete" the wizard. How the wizard is completed is up to the form designer. There is no built-in completion step or action. The wizard header does not display a Finish button.

In most cases, you should design the last step of the wizard with a button that performs the desired action. The button may simply trigger a save-to-database, and/or the button may use a command such as Add Plan File or RunEvent.

This last step may also include some explanatory text with a summary of what the user is about to do. For example, the last step could contain text such as "Click the **Create Request** button to create a new capital request with the following properties", followed by a summary of the key properties. Once the button has been clicked, you may want to update the page to display confirmation text and information on what to do next. For example: "Your request has been created. You can access this request from your Axiom Software home page."

Design alternatives

The Wizard Panel component is intended to streamline the process of creating guided input forms. However, it is possible to manually create all of the functionality that a Wizard Panel provides.

For example, you can manually create several layers and use buttons to dynamically show and hide those layers as appropriate. The buttons can simulate the "Previous" and "Next" progression provided by the wizard, or they can be used as "tabs" to toggle among the available layers. Once all required inputs have been gathered, you can dynamically show a "Finish" or "OK" button that triggers a particular action, and then display a final confirmation screen. In this case, you must set up all of the dynamic relationships yourself, so that the appropriate content displays for each screen.

Using a formatted grid for Wizard Panel content

When you place a Wizard Panel component on the form canvas, a Formatted Grid component is automatically added to the canvas at the same time. The default behavior of the Wizard Panel component assumes that you will use this grid to present the contents of each step.

When using this option, the `[Value]` column of the data source should contain Grid data source names. As the user moves through each step, the grid will be updated to use the specified data source for the current step.

Thematic formatted grids support a variety of user input controls, such as combo boxes, check boxes, text boxes, and buttons. Therefore it is possible to create very robust step contents using just this single component type. Using a grid is a good choice when your step contents are very structured—for example, using a column for labels and a column for inputs—and you need all steps in the wizard to be structured consistently.

Formatted Grid configuration

The default Formatted Grid component is configured as follows:

- The **Grid Formatting** is set to **Thematic**.
- The component-level **Style** is set to **docked-to-container**. This style positions the grid within the wizard and configures the size to fill the screen. It is recommended to leave this style as is.

  NOTE: If you are using a legacy skin (any skin except Axiom2018), then the component-level style is set to **wizardpanel-content**. This style serves the same purpose as the docked-to-container style, but uses a different name that is designed to work with the legacy Wizard theme. If you convert an existing form to use the Axiom2018 skin, you must change the style on the grid to use **docked-to-container** instead.

- The **Data Source** is set to `[WizardPanel1.SelectedValue]`. This is special syntax that causes the grid to get the data source name from the **Selected Value** field of the Wizard Panel component (which by default is named WizardPanel1). This means that you do not have to set up this field with a formula. Note the following:
  - If you give the Wizard Panel a different name, you must update this syntax for the new name.
  - The Formatted Grid component will continue to show with a "missing data source" error until you populate the Selected Value field of the Wizard Panel component with the value for the first step (which must be a valid data source name).
- All other Formatted Grid component properties can be set as desired.

Displaying wizard content in the grid

To use the grid to display the wizard content, you should do the following:

- Create a Grid data source for each step of the wizard, and design each data source with the content necessary for each step. You may find it easiest to place each data source on its own sheet, and give each sheet and data source the same name, such as Step1. If the number of steps in the wizard changes dynamically, then you may prefer to use content-oriented names instead (for example, UserInfo for a step that gathers information about the user).
- In the `[Value]` column of the WizardPanel data source, enter the Grid data source name that corresponds to each step, using the syntax *SheetName*!*DataSourceName*. For example, the first step might use a data source name of `Step1!Step1`.
- In the **Selected Value** field of the Wizard Panel component, enter the value for the first step of the wizard. For example, `Step1!Step1`.

  This is not technically required, as the Wizard Panel will use the first step by default when the form is rendered. However, if the Selected Value field is left blank, then the Formatted Grid component will display with an error in the Form Designer, because it does not have an assigned data source. You may want to populate the Selected Value field just to resolve this error while you are working in the Form Designer.

Assuming that the Formatted Grid component is using the default `[WizardPanel1.SelectedValue]` syntax for the data source, this setup will cause the grid to dynamically change data sources as the user moves through each step.

When a user views the form, the Selected Value of the Wizard Panel component starts with the data source name of the first step. Therefore the Formatted Grid component will display the contents of that data source. As the user moves through different steps in the wizard, the Selected Value field will update with the appropriate data source name of each step, and the contents of the grid will also update accordingly.

Using panels for Wizard Panel content

As an alternative to the default Formatted Grid component, you can use a series of Panel components to define the contents of each step in the wizard. Each step in the wizard should have a corresponding panel that contains the child components used to gather the necessary user inputs for that step, or to present information for the step.

When using this option, the `[Value]` column of the data source should contain Panel component names. As the user moves through each step, the panel for the current step will be automatically visible, and the rest of the panels will be hidden. This behavior is automatic and does not require any special setup.

The Panel components used for each step can contain any form component. This option provides full flexibility in defining the step content.

Panel configuration

The first step in using panels is to delete the default Formatted Grid component from the parent Wizard Panel component. Then, you can add Panel components as needed. Note the following when adding panels:

- When you drag and drop a Panel component onto the Wizard Panel component, the **Style** is automatically set to **docked-to-container**. This style positions the grid within the wizard and configures the size to fill the screen. It is recommended to leave this style as is.

  NOTE: If you are using a legacy skin (any skin except Axiom2018), then the component-level style is set to **wizardpanel-content**. This style serves the same purpose as the docked-to-container style, but uses a different name that is designed to work with the legacy Wizard theme. If you convert an existing form to use the Axiom2018 skin, you must change the style on the panel to use **docked-to-container** instead.

- It is recommended to create a layer for each panel used in the wizard, so that you can show only one panel at a time in the Form Designer. Otherwise, you will have to continually use **Send to Back** / **Bring to Front** to work with each panel. Additionally, having multiple overlapping panels visible in the designer at the same time can cause confusion when attempting to add components to the form, as components may not be assigned as you expect. These layers are not used to determine visibility of components when the form is rendered, the layers are only to make it easier to work with the panels in the Form Designer.
- After adding the first panel to the canvas, you should create a layer for that panel and then hide it in the designer before adding the next panel. If you do not hide the panel, then you must be careful when dragging the next panel onto the canvas to make sure it is added as a child of the Wizard Panel component instead of as child of the other Panel component. If the other Panel component is hidden, then the new panel will automatically be added as a child of the Wizard Panel component.

Panels should be configured as follows:

- The **Component Name** should be changed to a relevant name for the panel contents. You might use step-specific names such as PanelStep1 and PanelStep2, or content-specific names such as PanelUserInfo and PanelRequestDetails.
- All other Panel component properties can be set as desired.

To add contents to each panel, you should hide all panel layers except for the layer that contains the panel that you want to work with (assuming you are using layers as recommended previously). Make sure that all components are fully placed within the panel so that they are added as child components of the panel. It will be obvious if you accidentally place a component within the Wizard Panel component instead, as it will immediately be sized and positioned to fill the screen, whereas child components of the regular panel will behave as normal.

Make sure the panel is assigned to its intended layer before you start adding child components to the panel. When you add child components to the panel, they will automatically be assigned to the same

layer to start. But if you later change the layer of the panel, the child components will not be similarly updated.

Displaying wizard content using panels

To use panels to display the wizard content, you should do the following:

- Create a Panel component for each step of the wizard, and configure them as described in the previous section.
- In the `[Value]` column of the WizardPanel data source, enter the Panel component name that corresponds to each step. For example, the first step might use a component name of PanelStep1.
- In the **Selected Value** field of the Wizard Panel component, enter the value for the first step of the wizard. For example, `PanelStep1`.

  This is optional, as the Wizard Panel will use the value for the first step of the wizard by default when the form is viewed.

When a user views the form, the Selected Value of the Wizard Panel component starts with the Panel component name for the first step. This panel is automatically visible in the form, and the rest of the panels are automatically hidden. As the user moves through different steps in the wizard, the Selected Value field will update with the appropriate component name for each step, and the panels will show and hide accordingly.

This dynamic visibility happens automatically and does not require any configuration on the panels. It is not necessary to set up the Visible property of each Panel component (or its layer) with a formula. Note that the dynamic visibility is handled by Axiom Software in the background, and does not affect the Visible property of the panels. This means that if you view the form and download the source file from the wizard, the Visible properties of the panels on the Form Control Sheet will remain unchanged, even though only the panel for the current step is visible in the form.

Using Charts

The chart components provide graphing and charting capabilities for dashboards. In the Form Designer, chart components are available in the **Charts** section along the left-hand side of the screen.

- Area Chart: Display data in an area chart.
- Bar Chart: Display data in a horizontal bar chart.
- Bubble Chart: Display multidimensional data in a bubble chart.
- Bullet Chart: Display a current value and a target value along a defined measurement scale.
- Column Chart: Display data in a vertical column chart.
- Hierarchy Chart: Display data in an expandable and collapsible hierarchy.
- KPI Panel: Display one or more KPI values.
- Line Chart: Display data in a line chart.
- Linear Gauge: Display a value along a defined measurement scale, with the scale presented in a linear format.
- Map View: Display geospatial data on a map view.
- Pie Chart: Display data using a pie chart.
- Scatter Chart: Display multidimensional data in a scatter point chart.
- Scatter Line Chart: Display multidimensional data in a scatter line chart.
- Sparkline Chart: Display trend data in a simple at-a-glance chart.
- Radial Gauge: Display a value along a defined measurement scale, with the scale presented in a radial format.
- Waterfall Chart: Display data in a waterfall chart.

NOTE: Your Axiom Software license determines whether you have access to chart components. For more information, see Licensing requirements for Axiom forms.

Area Chart component

Area Chart components display information as a series of data points that are connected and shaded to form an area. An area chart is a variation of a line chart, where the area below the line is shaded to form a distinct shape. The areas can be stacked or overlapping. Area charts are part of the XYChart family, which

includes bar, line, and column charts. All of these charts use the same data source type (XYChart) and have the same basic component properties.

Defining an area chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of an Area Chart component on the Axiom form canvas.

Area charts can also support interactivity, to change the contents of the Axiom form based on the currently selected data point in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Area Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

`[XYChart;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[Series]`

Each row flagged with this tag defines a series of data to be displayed in the chart. Each series will use a different color.

`[XValueName]`

This row contains the names of each XValue column in the chart. These names will display along the primary axis of the chart (the X-axis for most charts; the Y-axis for bar charts).

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. These names will be displayed in the chart legend, if the chart is configured to show a legend (as defined in the component settings).

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`. If omitted, then all series in the chart will use the Default Series Kind as defined in the component settings. If a data source contains multiple kinds of series then it is known as a combination chart (for example, one or more column series combined with a line series).

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[Axis]

Optional. Specifies the Y-axis scale for each series. This column is only required if the chart has both a primary and secondary Y-axis. If omitted, the primary Y-axis scale is assumed. See Using two Y-axis scales with combination XYCharts.

[VisibleinLegend]

Optional. Specifies whether a particular series is shown in the chart legend (True/False). If omitted, all series are shown. This setting only applies if the chart is configured to show a legend (as defined in the component settings).

NOTES:

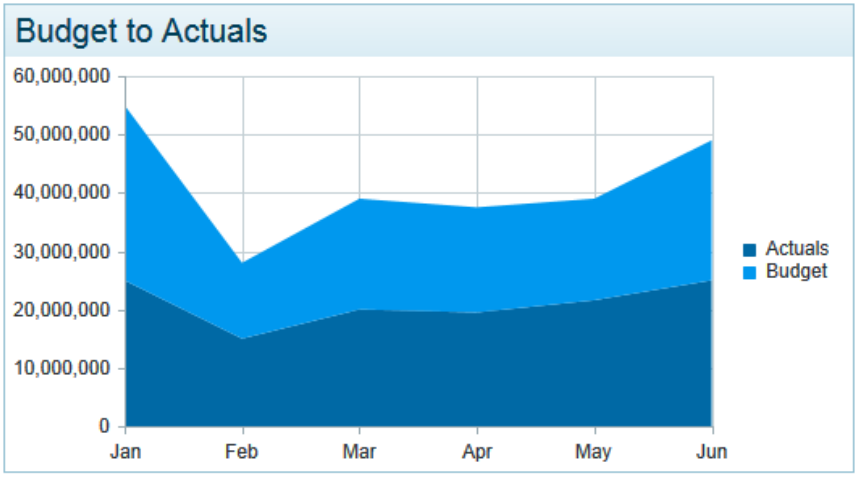
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple actual-to-budget data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J
1										
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Area	Actuals	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Area	Budget	30000000	13000000	19000000	18000000	17400000	24000000
6										

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Area Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight D3:J5) and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows (using the default behavior of stacked series for area charts):



Component properties

You can define the following properties for an Area Chart component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.

Item	Description
Selected Label Selected Series	The currently selected data point in the chart. This is identified by the corresponding label for the data point (the XValueName) and the Series that the data point belongs to. These settings are only used if the chart is configured to support interactivity. These settings serve two purposes: • They specify the initially selected data point of the chart, when the user first opens the form. You can leave the settings blank to have no initial selection, or you can enter an XValueName from the data source into the Selected Label field, and the corresponding Series name into the Selected Series field. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference these settings. • When a user views the form and selects a data point in the chart, the XValueName and Series name of the selected point will be submitted back to the source file and placed in these cells on the Form Control Sheet. Formulas can reference these cells in order to dynamically change the form based on the currently selected data point in the chart.
Auto Submit	Specifies whether the Axiom form is automatically refreshed when a user selects a data point in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will refresh unnecessarily if the user clicks on data points in the chart. If enabled, then the form automatically refreshes when the user selects a data point in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.

Item	Description
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Column Chart on the canvas, then the default is automatically set to Column. You can change the default chart type by changing this value.
Composition Kind	Specifies the composition of items in the chart when multiple series are present: • **Side by Side** (default for Bar, Column, and Line Chart components): Bars, columns, and lines for each series are displayed side-by-side. For area charts, the areas overlap the same space. **NOTE:** If you choose this option for an area chart, you may also want to set the **Area Series Opacity** to **Translucent**, so that you can see the detail for overlapping areas. • **Stacked** (default for Area Chart components): Series are stacked in a single bar or column. For area and line charts, each area or line is stacked on top of each other. The selected composition kind applies to all series in the chart.

Item	Description
Area Series Opacity	Specifies the opacity of area series within the chart: • **Opaque** (default): Area series are opaque. • **Translucent**: Area series are translucent. This is typically selected if the **Composition Kind** of the chart is set to **Side by Side**, so that you can see all areas in the chart. This setting is ignored for all other series kinds.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the XValueNames defined in the data source, and the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Name Rotation	The degree of rotation for the chart names (the XValueNames from the data source). By default this is blank, which means that the names are not rotated. To rotate the names, enter a value from -360 to 360. The purpose of this setting is to allow displaying longer names as vertical or slanted. For example, a value of -45 displays the name as slanted upward, whereas a value of 45 displays the name as slanted downward.  *-45 degree name rotation*  *45 degree name rotation*
Primary Y-Axis Label	Optional. The label for the primary Y-axis. This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Primary Y-Axis Format	Optional. Specifies the format for the primary Y-axis values: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Primary Y-Axis Decimals	Optional. Specifies how many decimal places to show on the primary Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Min Primary Y-Axis Max	Optional. Specifies the maximum value and the minimum value for the primary Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Primary Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
Use Secondary Y-Axis	Select this option if you want to create a chart with two different Y-axis scales. If this check box is selected, then another series of Y-Axis settings will display for the Secondary Y-Axis. These settings work the same way as the settings for the Primary Y-Axis. Typically, multiple Y-axis scales are only used with combination charts, meaning charts with two types of series. For more information, see Creating combination charts.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the XYChart family. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Area Chart component can be set up to allow the user to select an area in the chart, and then submit the selected point on the area back to the source file. The selected point is written to the **Selected Label** and **Selected Series** settings on the Form Control Sheet, using the XValueName and the corresponding Series name of the selected area.

NOTES:

- If the chart areas are overlapping instead of stacking, then it may be difficult for users to select particular Series at points where the areas overlap.
- Selecting particular XValues in an area chart is difficult, because there is no distinct beginning and end to the XValue (unlike in a column chart, where the user can select a distinct column shape). For example, if the user wants to select the XValue for February for a particular Series, they could unintentionally select the value for January or March depending on where they click on the area.

If you want the Axiom form to respond to the currently selected area, then you must set up the file so that another component references the selected label and/or series and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular area in the chart.

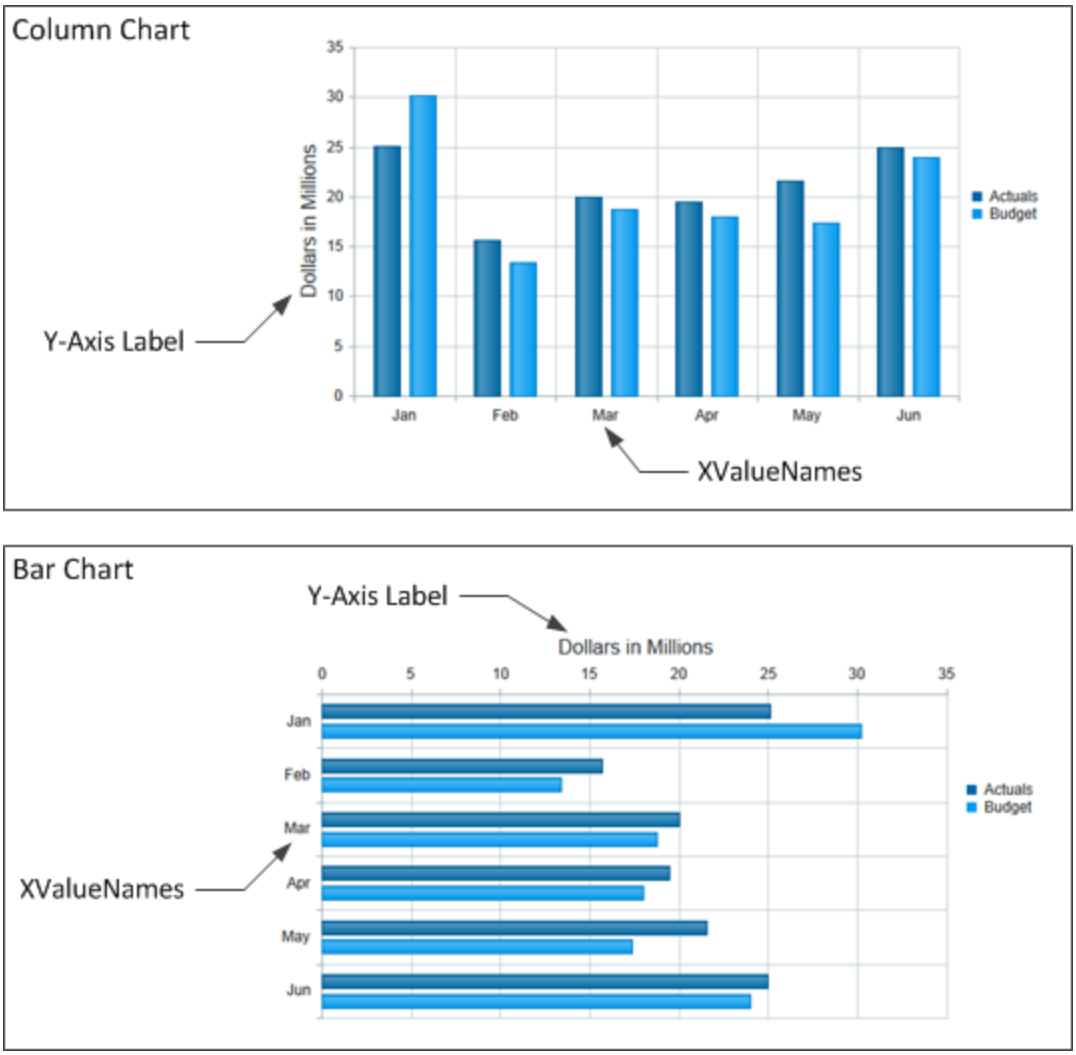
Example

The Axiom form could contain an area chart that shows budget and actuals data by month. If you want users to be able to see the details about the data in any particular month, you could set up a second chart that references the selected label and series of the first chart. For example, if the user selects the Budget area for February in the first chart, the second chart will be updated to show detailed budget data for February. The second chart could support additional interactivity so that the Axiom form user can decide how they want to view this detailed budget data (for example, broken out by account category or by department regions).

Bar Chart component

Bar Chart components display information using vertical bars. The series of data represented by the bars can be displayed side-by-side or stacked. Bar charts are part of the XYChart family, which includes column, line, and area charts. All of these charts use the same data source type (XYChart) and have the same basic component properties.

Bar charts are different from other XYChart kinds in that the orientation of the chart axes is flipped. For column, line, and area charts, the XValues defined for the chart are displayed horizontally across the X-axis, while the scale values are displayed vertically down the Y-axis. Bar charts use the opposite configuration—XValues are displayed vertically down the Y-axis, and scale values are displayed horizontally across the Y-axis. However, the data source tags and property names remain the same for a bar chart, so keep in mind the flipped orientation when defining these settings.



Example of orientation differences

Defining a bar chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Bar Chart component on the Axiom form canvas.

Bar charts can also support interactivity, to change the contents of the Axiom form based on the currently selected data point in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Bar Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each series will use a different color.

[XValueName]

This row contains the names of each XValue column in the chart. These names will display along the primary axis of the chart (the X-axis for most charts; the Y-axis for bar charts).

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. These names will be displayed in the chart legend, if the chart is configured to show a legend (as defined in the component settings).

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`. If omitted, then all series in the chart will use the Default Series Kind as defined in the component settings. If a data source contains multiple kinds of series then it is known as a combination chart (for example, one or more column series combined with a line series).

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[Axis]

Optional. Specifies the Y-axis scale for each series. This column is only required if the chart has both a primary and secondary Y-axis. If omitted, the primary Y-axis scale is assumed. See Using two Y-axis scales with combination XYCharts.

[VisibleinLegend]

Optional. Specifies whether a particular series is shown in the chart legend (True/False). If omitted, all series are shown. This setting only applies if the chart is configured to show a legend (as defined in the component settings).

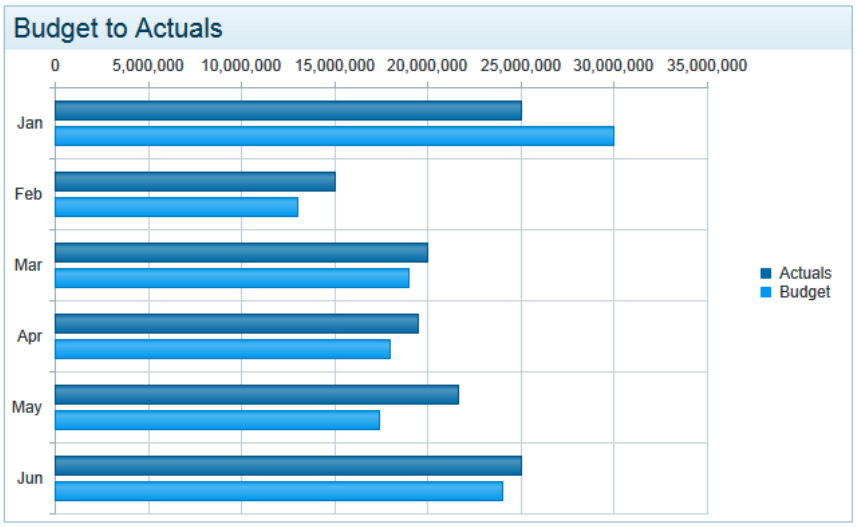
NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple actual-to-budget data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J
1										
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Bar	Actuals	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Bar	Budget	30000000	13000000	19000000	18000000	17400000	24000000

The resulting chart would appear as follows:



To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Bar Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight D3:J5) and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

Component properties

You can define the following properties for a Bar Chart component:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Selected Label Selected Series	The currently selected data point in the chart. This is identified by the corresponding label for the data point (the XValueName) and the Series that the data point belongs to. These settings are only used if the chart is configured to support interactivity. These settings serve two purposes: • They specify the initially selected data point of the chart, when the user first opens the form. You can leave the settings blank to have no initial selection, or you can enter an XValueName from the data source into the Selected Label field, and the corresponding Series name into the Selected Series field. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference these settings. • When a user views the form and selects a data point in the chart, the XValueName and Series name of the selected point will be submitted back to the source file and placed in these cells on the Form Control Sheet. Formulas can reference these cells in order to dynamically change the form based on the currently selected data point in the chart.
Auto Submit	Specifies whether the Axiom form is automatically refreshed when a user selects a data point in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will refresh unnecessarily if the user clicks on data points in the chart. If enabled, then the form automatically refreshes when the user selects a data point in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.

Item	Description
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Column Chart on the canvas, then the default is automatically set to Column. You can change the default chart type by changing this value.
Composition Kind	Specifies the composition of items in the chart when multiple series are present: • **Side by Side** (default for Bar, Column, and Line Chart components): Bars, columns, and lines for each series are displayed side-by-side. For area charts, the areas overlap the same space. • **Stacked** (default for Area Chart components): Series are stacked in a single bar or column. For area and line charts, each area or line is stacked on top of each other. The selected composition kind applies to all series in the chart.

Item	Description
Area Series Opacity	Specifies the opacity of area series within the chart: • **Opaque** (default): Area series are opaque. • **Translucent**: Area series are translucent. This is typically selected if the **Composition Kind** of the chart is set to **Side by Side**, so that you can see all areas in the chart. This setting is ignored for all other series kinds.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the XValueNames defined in the data source, and the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Name Rotation	The degree of rotation for the chart names (the XValueNames from the data source). By default this is blank, which means that the names are not rotated. To rotate the names, enter a value from -360 to 360. The purpose of this setting is to allow displaying longer names as vertical or slanted. For example, a value of -45 displays the name as slanted upward, whereas a value of 45 displays the name as slanted downward.  *-45 degree name rotation*  *45 degree name rotation* **NOTE:** For bar charts, the names run down the Y-axis instead of along the X-axis as shown here.
Primary Y-Axis Label	Optional. The label for the primary Y-axis. This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions". **NOTE:** If the chart is a bar chart, then the axes are flipped. XValueNames from the data source are displayed along the traditional Y-axis (down the side of the chart), whereas Y-axis labels are displayed along the traditional X-axis (across the width of the chart).

Item	Description
Primary Y-Axis Format	Optional. Specifies the format for the primary Y-axis values: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Decimals	Optional. Specifies how many decimal places to show on the primary Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Min Primary Y-Axis Max	Optional. Specifies the maximum value and the minimum value for the primary Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Primary Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.

Item	Description
Use Secondary Y-Axis	Select this option if you want to create a chart with two different Y-axis scales. If this check box is selected, then another series of Y-Axis settings will display for the Secondary Y-Axis. These settings work the same way as the settings for the Primary Y-Axis. Typically, multiple Y-axis scales are only used with combination charts, meaning charts with two types of series. For more information, see Creating combination charts.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the XYChart family. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Bar Chart component can be set up to allow the user to select a bar in the chart. The selected bar is submitted back to the source file, and written to the **Selected Label** and **Selected Series** settings on the Form Control Sheet, using the XValueName and the corresponding Series name of the selected bar.

If you want the Axiom form to respond to the currently selected bar, then you must set up the file so that another component references the selected label and/or series and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular bar in the chart.

Example

The Axiom form could contain a bar chart that shows budget and actuals data by month. If you want users to be able to see the details about the data in any particular month, you could set up a second chart that references the selected label and series of the first chart. For example, if the user selects the Budget bar for February in the first chart, the second chart will be updated to show detailed budget data for February. The second chart could support additional interactivity so that the Axiom form user can decide how they want to view this detailed budget data (for example, broken out by account category or by department regions).

Bubble Chart component

The Bubble Chart component illustrates data with three variables using a collection of points, where one variable determines a data point's position on the horizontal axis, and a second variable determines the data point's position on the vertical axis. The third variable determines the relative size of the data point.

Bubble charts are part of the ScatterChart family, which includes scatter line and scatter charts. All of these charts use the same data source type (ScatterChart) and have the same basic component properties.

Defining a bubble chart is a two-part process that requires the following:

- Creation of a ScatterChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Bubble Chart component on the Axiom form canvas.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Bubble Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[ScatterChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a data point to be displayed in the chart. Multiple rows can belong to the same series, depending on the name entered in the `[SeriesName]` column.

Column tags

[XValue]

This column contains the values to determine the x-axis position (horizontal) of each data point.

[YValue]

This column contains the values to determine the y-axis position (vertical) of each data point.

[Size]

This column determines the size of each data point (bubble) within the series. You can enter any relevant values for the data point; the bubble size in the chart will be rendered proportionally to the other size values in the series (meaning, the largest value will render as the largest bubble in the series, the next largest value will be a slightly smaller bubble, etc.). This column does not apply to Scatter Charts or Scatter Line Charts.

[Label]

Optional. This column contains the name of each individual data point in the chart. By default, the label will display in a tooltip when the user hovers over the data point. If labels are enabled for the chart, then the label will also display next to the data point within the chart. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[Color]

Optional. This column specifies the color assignment for each series or each data point. If omitted, then colors will be dynamically determined based on the style, theme, or skin (in that order). See Specifying chart colors. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

If you are specifying colors and you want the color to apply to the entire series, then only use one color entry per series (leaving the rest blank), or repeat the same color entry on all items of the series.

Column tags (optional, series-wide)

All of the following tags are optional and apply to the entire series, not just the current data point. If you use any of these series-wide tags, you should make sure that each entry in the tag is the same for all data points that belong to the same series. If different entries are found within the same series, the first entry is used.

[SeriesName]

This column contains the names of each series in the chart. Multiple data points in the chart can belong to the same series by entering the same series name in this column. If this column is omitted, then all data points in the chart belong to a single unnamed series.

[Kind]

This column indicates the kind of each series, either Scatter, ScatterLine, or Bubble. If omitted, then all series in the chart will use the Default Series Kind defined in the component settings.

[VisibleinLegend]

This column indicates whether a series is shown in the chart legend, if the legend is enabled. If omitted, all series are shown. Indicate True or False. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

NOTES:

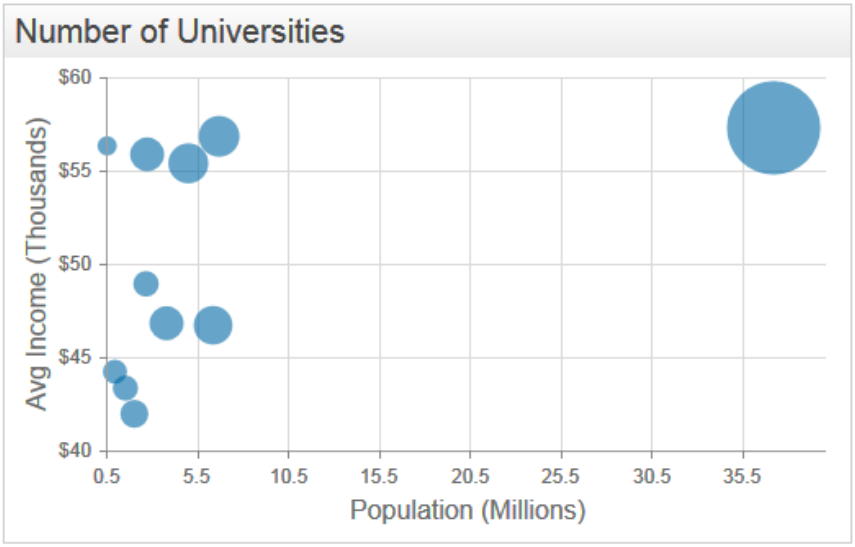
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows simple data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data. This example shows only the columns that are added by the Data Source Wizard; if you want to use the other columns then you must manually add them.

	A	B	C	D	E	F	G
1							
2		[ScatterChart;BubbleChartSource1]	[Kind]	[SeriesName]	[XValue]	[YValue]	[Size]
3		[Series]	Bubble	Series	6.392017	46.709	157
4		[Series]	Bubble	Series	37.253956	57.287	1079
5		[Series]	Bubble	Series	5.029196	55.387	170
6		[Series]	Bubble	Series	1.567582	43.341	48
7		[Series]	Bubble	Series	0.989415	44.222	41

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Bubble Chart**. If the data already exists in the sheet, you can first highlight the labels and data and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows:



NOTE: The labels for the X-Axis and Y-Axis ("Population" and "Avg Income") are defined in the component properties, not in the data source. It is also recommended to use the optional `[Label]` column to define a label for each bubble. You can choose to display the labels next to each bubble in the chart, or have them only display in tooltips when hovering the cursor over the bubble.

Component properties

You can define the following properties for a Bubble Chart component:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.

Item	Description
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. Select either **Bubble**, **Scatter**, or **Scatter Line**. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Bubble Chart on the canvas, then the default is automatically set to Bubble.
Show Labels	Specifies whether labels will display next to each data point in the chart. Labels are defined in the optional [Label] column within the data source. If your chart has many data points, you may want to disable this setting to avoid clutter in the chart. The labels will still display in tooltips when the user hovers over the data point, if labels are defined in the data source.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.

Item	Description
Y-Axis Label	Optional. Enter a label for the Y-axis (vertical). This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Y-Axis Format	Optional. Specify the format for the Y-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Y-Axis Decimals	Optional. Specify how many decimal places to show on the Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Y-Axis Min Y-Axis Max	Optional. Specify the maximum value and the minimum value for the Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.

Item	Description
Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
X-Axis Label	Optional. Enter a label for the X-axis (horizontal). This will display next to the X-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
X-Axis Format	Optional. Specify the format for the X-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
X-Axis Decimals	Optional. Specify how many decimal places to show on the X-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
X-Axis Min X-Axis Max	Optional. Specify the maximum value and the minimum value for the X-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
X-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the X-axis. By default, no scale is applied. Enter a number to scale all X-axis numbers by that value. The X-axis numbers will be divided by the specified value. For example, if an X-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the X-axis scale, not 35.
Bubble Units	If the chart is a Bubble Chart, specify a label for the bubble sizes. This label will display on the tooltip when a user hovers their cursor over a bubble. For example, if the bubble sizes represent the number of beds in a hospital, you might enter "Beds" as the bubble units. The tooltip would then display text such as "500 Beds".
Bubble Size Format	If the chart is a Bubble Chart, specify the format of the bubble sizes: **Number** (default), **Currency**, or **Percent**. This will impact the display of the size value in the tooltip when a user hovers their cursor over the bubble.
Bubble Size Decimals	If the chart is a Bubble Chart, specify how many decimal places to show for the bubble sizes. By default, no decimal places are shown (0). This will impact the display of the size value in the tooltip when a user hovers their cursor over the bubble.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the ScatterChart family. Only the generic styles are available.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Bullet Chart component

The Bullet Chart component for Axiom forms displays a value along a defined standard of measurement, comparing it against a target value. The chart is shaded to indicate where the current value falls within one of three ranges of performance, such as poor, satisfactory, and good.

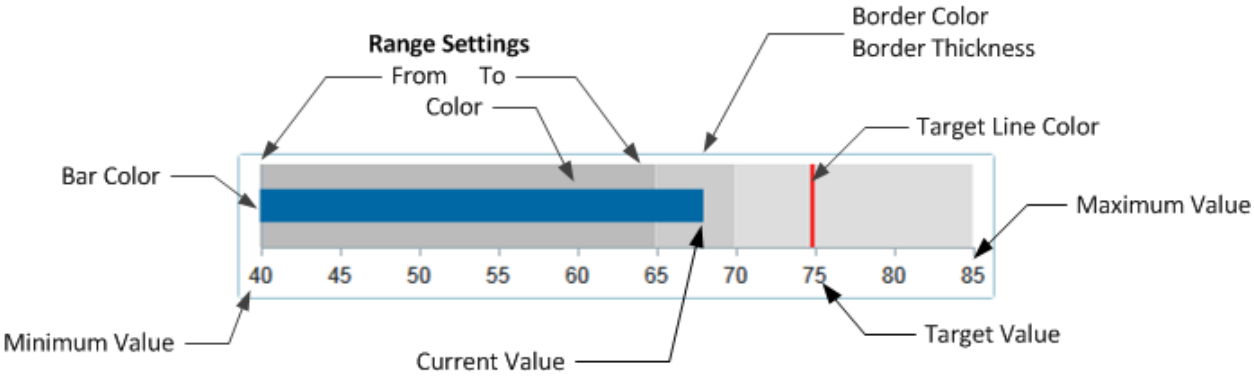
Bullet charts and gauges are visually similar, but are typically used for different purposes. Although both components display a value along a defined measurement scale, the bullet chart adds the concept of a target value and therefore explicitly communicates performance against a defined goal. The overall appearance of bullet charts is also more streamlined than gauges, which are often styled to resemble real-life measurement tools such as thermometers or speedometers.

NOTES:

- Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.
- The Bullet Chart component does not use the BulletChart data source; you must define all data for the chart within the component properties. Currently, the BulletChart data source is only for use with the Sparkline content tag in formatted grids. For more information see Using sparklines in Formatted Grids.

Bullet Chart properties

You can define the following properties for a Bullet Chart component. The following screenshot shows an example chart with the major properties that impact the display:



Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Current Value	The current value for the bullet chart. The current value is indicated by the chart bar. In most cases you will want to read the value from some location in the spreadsheet rather than typing a value into the component properties. You can use the Form Control Sheet to enter a formula for this property.
Target Value	The target value for the bullet chart. This value should represent the goal of whatever is being measured by the bullet chart. The target value is indicated by a colored line in the chart.
Minimum Value	The minimum value for the chart scale of measurement. By default this is 0.
Maximum Value	The maximum value for the chart scale of measurement. By default this is 100.
Bar Color	The color of the bar that indicates the current value. If left blank, the color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

Item	Description
Target Line Color	The color of the line that indicates the target value. If left blank, the color is determined by the style or skin (in that order). Color can be specified in the same way as described above for bar color.
Orientation	Specifies the orientation of the chart: **Vertical** or **Horizontal** (default).
Show Axis	Specifies whether axis labels display on the chart. By default this is enabled. If disabled, users can still view the chart values in the tooltip.

NOTE: Bullet charts do not have a Title property. If you want to display a title for the chart you must use a separate Label component.

Range properties

You can define up to three ranges for the bullet chart. Ranges are defined by a starting and ending value, and a color to shade that range. If you do not want to use a particular range, leave the settings for that range blank.

If at least one range is defined, then any measurement values on the chart that do not fall within a defined range are not colored. If you want the ranges to be continuous, then the **To** value of one range and the **From** value of the next range should be the same number. For example, if range one is from 0 to 20, then the from value for range two should be 20.

Range colors may be inherited from the style, theme, or skin, or may be manually specified. Colors can be manually specified in the same way as described above for bar color.

Item	Description
Range 1 From	The starting value of the range.
Range 1 To	The ending value of the range.
Range 1 Color	The color for the range.
Range 2 From	The starting value of the range.
Range 2 To	The ending value of the range.
Range 2 Color	The color for the range.
Range 3 From	The starting value of the range.
Range 3 To	The ending value of the range.
Range 3 Color	The color for the range.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Even though Bar Color, Target Line Color, and range colors can be affected by styles, these properties are exposed as component behavior properties because they are unique to this component type. Also, Axiom Software does not currently provide any styles specifically for bullet charts.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Column Chart component

Column Chart components provide a visual presentation of categorical data, using vertical columns to display each category's value along a defined scale. Column charts are part of the XYChart family, which includes bar, line, and area charts. All of these charts use the same data source type (XYChart) and have the same basic component properties.

Defining a column chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Column Chart component on the Axiom form canvas.

Column charts can also support interactivity, to change the contents of the Axiom form based on the currently selected data point in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Column Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each series will use a different color.

[XValueName]

This row contains the names of each XValue column in the chart. These names will display along the primary axis of the chart (the X-axis for most charts; the Y-axis for bar charts).

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. These names will be displayed in the chart legend, if the chart is configured to show a legend (as defined in the component settings).

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`. If omitted, then all series in the chart will use the Default Series Kind as defined in the component settings. If a data source contains multiple kinds of series then it is known as a combination chart (for example, one or more column series combined with a line series).

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[Axis]

Optional. Specifies the Y-axis scale for each series. This column is only required if the chart has both a primary and secondary Y-axis. If omitted, the primary Y-axis scale is assumed. See Using two Y-axis scales with combination XYCharts.

[VisibleinLegend]

Optional. Specifies whether a particular series is shown in the chart legend (True/False). If omitted, all series are shown. This setting only applies if the chart is configured to show a legend (as defined in the component settings).

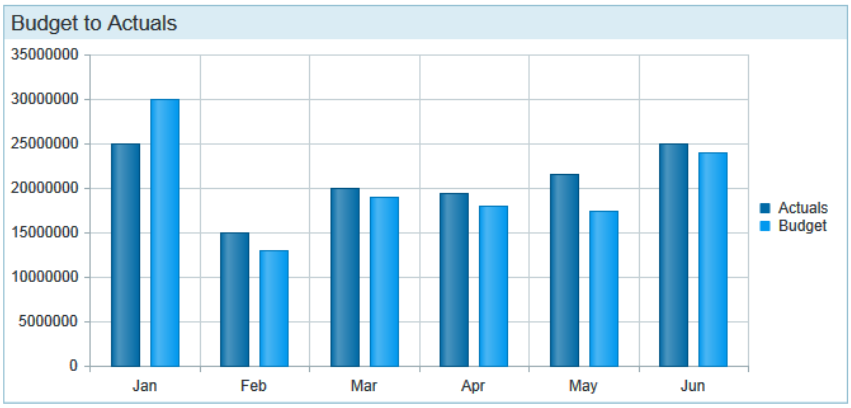
NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple actual-to-budget data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J
1										
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Column	Actuals	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Column	Budget	30000000	13000000	19000000	18000000	17400000	24000000

The resulting chart would appear as follows:



To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Column Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight D3:J5) and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

Component properties

You can define the following properties for a Column Chart component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Selected Label Selected Series	The currently selected data point in the chart. This is identified by the corresponding label for the data point (the XValueName) and the Series that the data point belongs to. These settings are only used if the chart is configured to support interactivity. These settings serve two purposes: • They specify the initially selected data point of the chart, when the user first opens the form. You can leave the settings blank to have no initial selection, or you can enter an XValueName from the data source into the Selected Label field, and the corresponding Series name into the Selected Series field. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference these settings. • When a user views the form and selects a data point in the chart, the XValueName and Series name of the selected point will be submitted back to the source file and placed in these cells on the Form Control Sheet. Formulas can reference these cells in order to dynamically change the form based on the currently selected data point in the chart.
Auto Submit	Specifies whether the Axiom form is automatically refreshed when a user selects a data point in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will refresh unnecessarily if the user clicks on data points in the chart. If enabled, then the form automatically refreshes when the user selects a data point in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.

Item	Description
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Column Chart on the canvas, then the default is automatically set to Column. You can change the default chart type by changing this value.
Composition Kind	Specifies the composition of items in the chart when multiple series are present: • **Side by Side** (default for Bar, Column, and Line Chart components): Bars, columns, and lines for each series are displayed side-by-side. For area charts, the areas overlap the same space. • **Stacked** (default for Area Chart components): Series are stacked in a single bar or column. For area and line charts, each area or line is stacked on top of each other. The selected composition kind applies to all series in the chart.

Item	Description
Area Series Opacity	Specifies the opacity of area series within the chart: • **Opaque** (default): Area series are opaque. • **Translucent**: Area series are translucent. This is typically selected if the **Composition Kind** of the chart is set to **Side by Side**, so that you can see all areas in the chart. This setting is ignored for all other series kinds.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the XValueNames defined in the data source, and the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Name Rotation	The degree of rotation for the chart names (the XValueNames from the data source). By default this is blank, which means that the names are not rotated. To rotate the names, enter a value from -360 to 360. The purpose of this setting is to allow displaying longer names as vertical or slanted. For example, a value of -45 displays the name as slanted upward, whereas a value of 45 displays the name as slanted downward.  *-45 degree name rotation*  *45 degree name rotation*
Primary Y-Axis Label	Optional. The label for the primary Y-axis. This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Primary Y-Axis Format	Optional. Specifies the format for the primary Y-axis values: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Primary Y-Axis Decimals	Optional. Specifies how many decimal places to show on the primary Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Min Primary Y-Axis Max	Optional. Specifies the maximum value and the minimum value for the primary Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Primary Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
Use Secondary Y-Axis	Select this option if you want to create a chart with two different Y-axis scales. If this check box is selected, then another series of Y-Axis settings will display for the Secondary Y-Axis. These settings work the same way as the settings for the Primary Y-Axis. Typically, multiple Y-axis scales are only used with combination charts, meaning charts with two types of series. For more information, see Creating combination charts.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the XYChart family. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Column Chart component can be set up to allow the user to select a column in the chart. The selected column is submitted back to the source file, and written to the **Selected Label** and **Selected Series** settings on the Form Control Sheet, using the XValueName and the corresponding Series name of the selected column.

If you want the Axiom form to respond to the currently selected column, then you must set up the file so that another component references the selected label and/or series and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular column in the chart.

Example

The Axiom form could contain a column chart that shows budget and actuals data by month. If you want users to be able to see the details about the data in any particular month, you could set up a second chart that references the selected label and series of the first chart. For example, if the user selects the Budget column for February in the first chart, the second chart will be updated to show detailed budget data for February. The second chart could support additional interactivity so that the Axiom form user can decide how they want to view this detailed budget data (for example, broken out by account category or by department regions).

Hierarchy Chart component

The Hierarchy Chart component can be used to present information in a hierarchical or tree view, where "parent" nodes link to one or more "child" nodes.

Defining a hierarchy chart is a three-part process that requires the following:

- Creation of a HierarchyChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Hierarchy Chart component on the Axiom form canvas.
- Creation of one or more Label templates to define the display of hierarchy nodes.

Hierarchy charts can also support form interactivity, to change the contents of the Axiom form based on the currently selected node in the hierarchy.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Hierarchy Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

`[HierarchyChart;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a Hierarchy Chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[Node]`

Each row flagged with this tag defines a node to display in the hierarchy chart.

Column tags

[ID]

This column identifies each node in the chart with an ID. The ID can be numeric or string, but it must be unique for each node.

[ParentID]

This column identifies the parent node for each individual node. The node that you want to display as the top-most node must be blank. All other nodes should be assigned parent ID that matches an ID in the [ID] column.

[Template]

Optional. This column indicates the Label template to use for each node. If omitted, or if left blank for a particular node, the default template defined for the Hierarchy Chart component will be used.

[Collapsed]

Optional. This column indicates whether the node should be collapsed or not when the chart is initially rendered (True or False). If omitted, all nodes will be expanded.

This column can also be used to record the current state of each node when the Axiom form is refreshed, so that the collapsed or expanded state can be honored during the refresh. The cells in this column must be unlocked in order to write back the current state. If this column is present but the cells are locked, then the original values in this column will be applied each time a refresh occurs. If this column is not present, then the current state will not be honored and all nodes will be expanded.

[Data.*Variable*]

Optional. These columns are user-definable and can contain any value that you want to pass back to the Label template. In most cases you will want to create at least one column such as `[Data.Text]` to define the text to display in the label for that node. By default, the wizard will automatically create tags for data columns based on the number of columns you have selected, using the name of the column. You can edit these tags as needed and/or create new tags.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

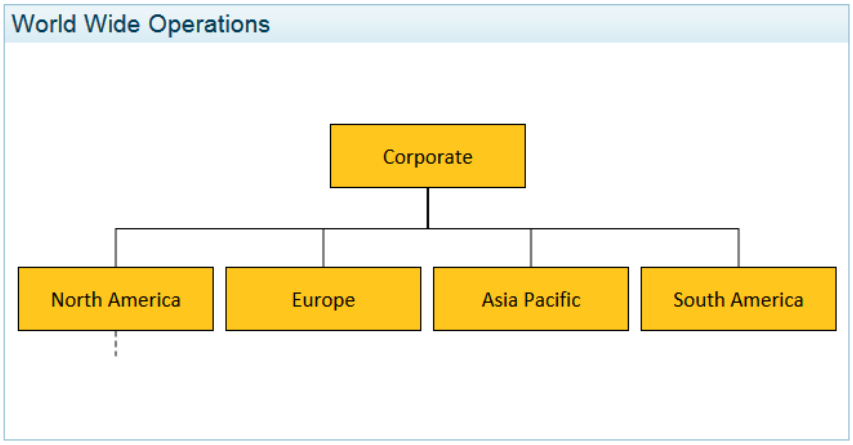
The following example shows a simple hierarchy flagged in a sheet. In real implementations this data might be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H
1								
2								
3		[HierarchyChart;Corporate]	[ID]	[ParentID]	[Template]	[Collapsed]	[Data.Text]	
4		[Node]	Corporate WorldWide		Label1	FALSE	Corporate WorldWide	
5		[Node]	North America	Corporate WorldWide	Label1	TRUE	North America	
6		[Node]	Europe	Corporate WorldWide	Label1	TRUE	Europe	
7		[Node]	Asia Pacific	Corporate WorldWide	Label1	TRUE	Asia Pacific	
8		[Node]	South America	Corporate WorldWide	Label1	TRUE	South America	
9		[Node]	United States	North America	Label2	FALSE	United States	
10		[Node]	Canada	North America	Label2	FALSE	Canada	
11		[Node]	Mexico	North America	Label2	FALSE	Mexico	

In this example we are using the node name as the ID and then simply referencing that name in the [Data.Text] column, but this is not required—you can use any unique value for the ID.

To use the Data Source Wizard to add the tags, right-click a cell and select **Create Axiom Form Data Source > Hierarchy Chart**. You can right-click a single empty cell to place the initial tags and then fill out the data, or you can have the data already in the spreadsheet and highlight the applicable data to add the tags. The cells in the row above the data and the column to the left of the data must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows:



NOTE: The dashed line under North America indicates the node is collapsed. The nodes underneath North America will not display until the node is expanded. See Collapse and expand behavior.

Component properties

You can define the following properties for an Hierarchy Chart component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.

Item	Description
Default Template	Optional. Specifies the default Label template to be used for any node that does not have an assigned template in the [Template] column of the data source. If the [Template] column is omitted, then a default template is required and will apply to all nodes. You can select from any Label component name that is defined in the Axiom form.
Orientation	The orientation of the hierarchy chart, either: • **Left Right** (default): The nodes in the chart expand horizontally, starting with the parent node on the left and working across to the right. • **Top Down**: The nodes in the chart expand vertically, starting with the parent node at the top and moving downward to the bottom.
Connector Style	The style of the connector lines between each parent and child node, either **Straight** or **Elbow**.
Connector Color	The color of the connector lines. If left at **Default**, the default color for the style or skin is used. Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

Item	Description
Selected Value	The selected value in the hierarchy chart. This setting serves two purposes: • It specifies the initially selected node in the hierarchy, when the user first opens the form. You can leave this blank to have no initial selection, or you can enter an ID from the `[ID]` column in the data source. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference this setting. • When a user views the form and selects a node in the hierarchy, the ID for the selected item will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected node. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected value to be read from and written to the specified cell reference instead of directly within the Selected Value cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected value is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Auto Submit	Specifies whether the Axiom form is automatically updated when a user changes the state of the component. By default, this is enabled, which means that the form automatically updates when the user selects a different node in the hierarchy. If this setting is disabled, then the user must use a Button component in order to update the form for the changed state.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Even though Connector Color can be affected by styles, this property is exposed as a component behavior property because it is unique to this component type. Also, Axiom Software does not currently provide any styles specifically for hierarchy charts.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Hierarchy Chart component allows the user to select a node in the hierarchy. This selected node is submitted back to the source file, and written to the **Selected Value** setting on the Form Control Sheet, using the value from the `[ID]` column for that node.

If you want the Axiom form to respond to the currently selected node, then you must set up the file so that another form component references the selected node and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

For example, you could have a hierarchy chart like the example in this topic, where each node represents a world region. Users could select different nodes in the hierarchy to impact the data displayed in the entire Axiom form, or just to change the data in a single component such as a column chart.

NOTE: Axiom Software does not apply any formatting to the selected node in the Axiom form to indicate to the user that the node is selected. If desired, you can set up the `[Template]` column for the data source to use a dynamic formula, so that a different Label template is used if the node is the currently selected item.

Defining Label templates

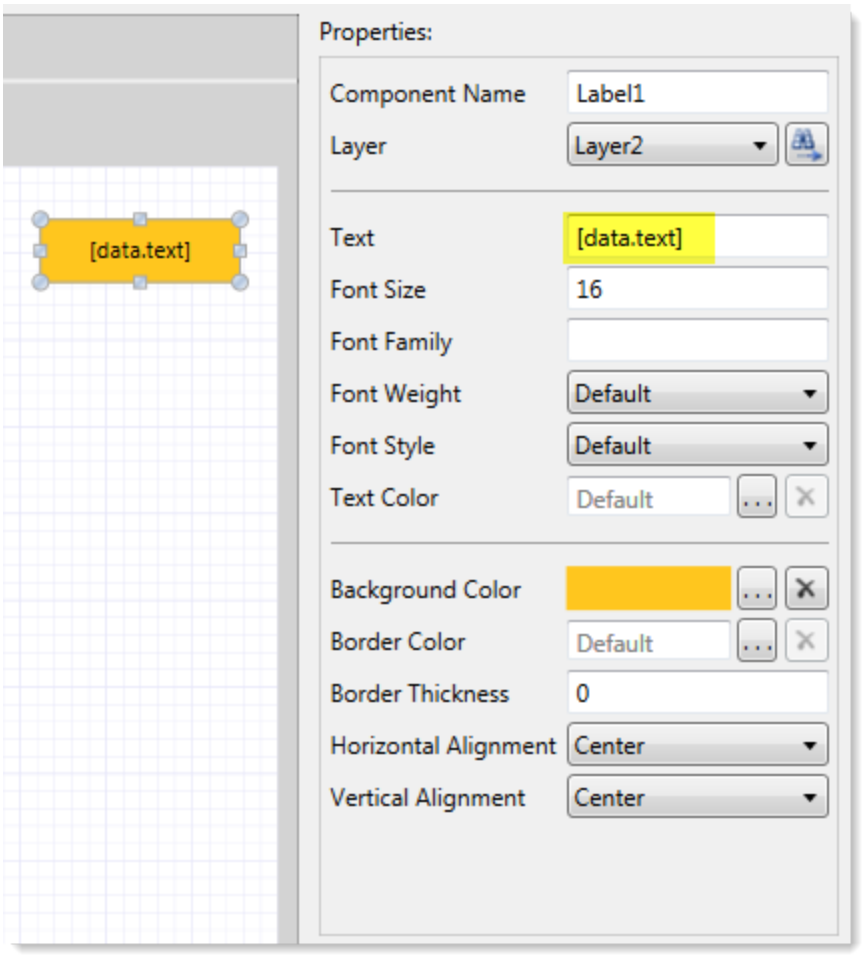
Hierarchy Chart components require one or more associated Label components to define how each node will be displayed in the chart. These Label templates should be hidden on the Axiom form, either by setting the Visible property for each individual label, or by placing all the labels on a layer and then setting the Visible property for that layer.

You can set a default Label template for the chart, which will be used for any node that does not have an assigned template within the `[Template]` column of the data source. If all nodes use the same template, then you can just use the default template and omit the `[Template]` column.

You can configure the Label templates as desired. There is no special setting on the Label component to indicate that it is a template for a Hierarchy Chart component; to use a Label component as a template, you specify the Label component name.

NOTE: By default, the height of a Label component is set to Auto. You must change this to specify an actual height. To do this, click **Show Advanced Settings** under the **Style** box, then adjust the **Height** property. For example, you can enter 30px to start, then further adjust the template on the canvas as needed.

Each node to use a Label template will inherit the configuration settings for the specified Label component, such as background color, border color, etc. You can use the `[Data.Variable]` columns to pass values into the template to use for that node. For example, at minimum you will want to define a column such as `[Data.Text]`, and then use that variable within the Text property of the Label component, so that the label displays the name of that node.

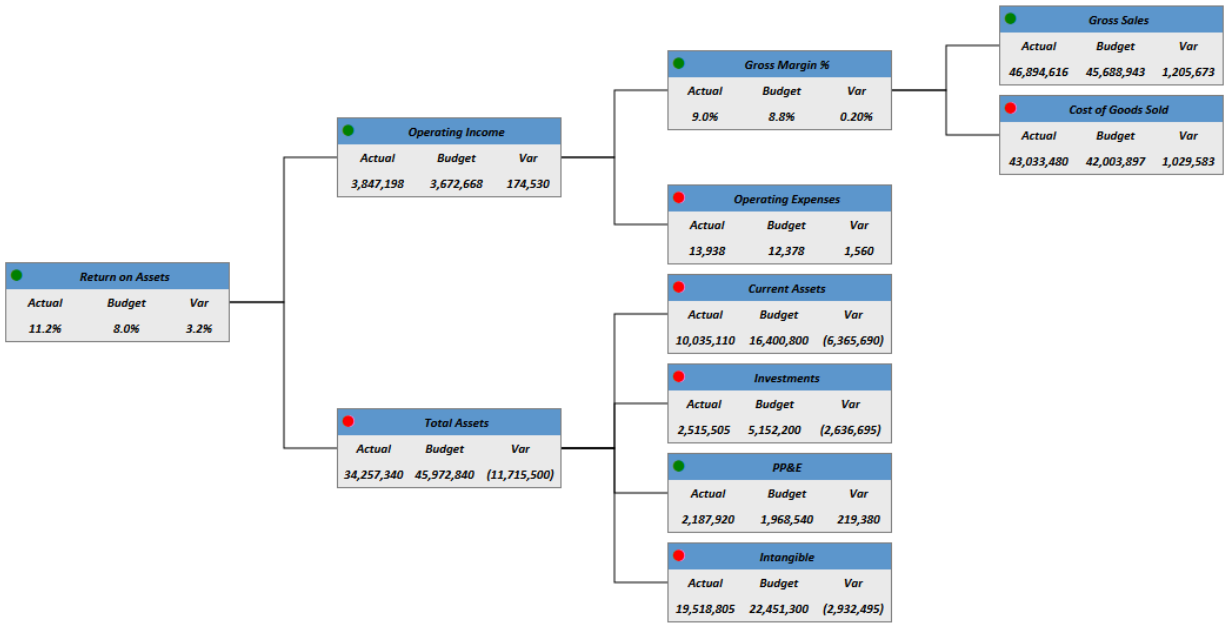


Example Label template

It is possible to use more sophisticated formatting for the label, such as presenting the text information on multiple rows and/or incorporating multiple colors and graphics. To do this you can enter any valid HTML into the Text property field (using the Form Control Sheet). The HTML entered into this field can reference any `[Data.Variable]` column, and can be used to control the display of the label.

IMPORTANT: The HTML option is only intended for advanced users who are familiar with HTML tagging. Axiom Software is not responsible for any HTML code entered into this field.

The following example shows a hierarchy chart created using this HTML tagging:



In this example, several columns of information are being passed to the label template, as well as special format tags to obtain the table formatting within the label.

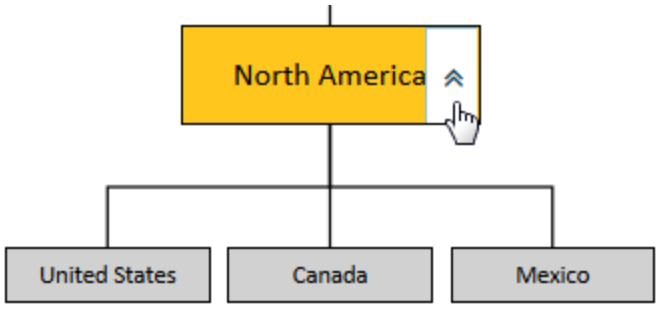
The specific sizing properties of the template—such as the Height and Width of the label, and the font size—are used to determine the desired proportions of the template, not the actual size. When rendered in the chart, the template will adjust as needed based on the number of items being shown and the space allotted for the hierarchy chart.

Collapse and expand behavior

Hierarchy nodes with child items can be collapsed and expanded within the Axiom form. To collapse or expand a node, you can hover your cursor over the left-hand side of the node. An arrow will display, indicating the item can be collapsed or expanded if the arrow is clicked.

If a node has a dashed line extending from it, this means the node has children and can be expanded by clicking the arrow.

Once the node has been expanded, it can be collapsed by clicking the arrow.



KPI Panel component

Using the KPI Panel component, you can display key performance indicators (KPIs) in a series of easily readable, eye-catching panels. The design of the component is flexible to accommodate various numbers and display configurations.

The KPI Panel component displays each KPI using a primary value or status, and several optional supporting values. Each KPI can be shown with an optional bullet chart or sparkline chart. KPIs can be flagged as trending up or down, which is indicated using an arrow and a color (green for up, red for down). KPIs can also be configured to execute one or more actions using a button in the top-right corner or a fly-out menu.

Defining a KPI panel is a two-part process that requires the following:

- Creation of KPI data using a predefined data structure. This data can be sourced from a KPI table or from a data source defined within the spreadsheet. Each KPI from the table or data source displays as its own box within the overall KPI panel.
- Placement and configuration of a KPI Panel component on the Axiom form canvas.

KPI Panel components can also support interactivity, to change the contents of the Axiom form based on the currently selected KPI box in the panel.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Component overview

KPI Panel components must point to a defined set of KPI data. This data can be provided using either of the following options:

- **Use a KPISource data source in the spreadsheet.** In this case, the component reads the KPI data from the spreadsheet. You can use any Axiom data query features to bring data into the spreadsheet and then perform calculations on that data as needed to arrive at the KPI data for the

data source.

OR

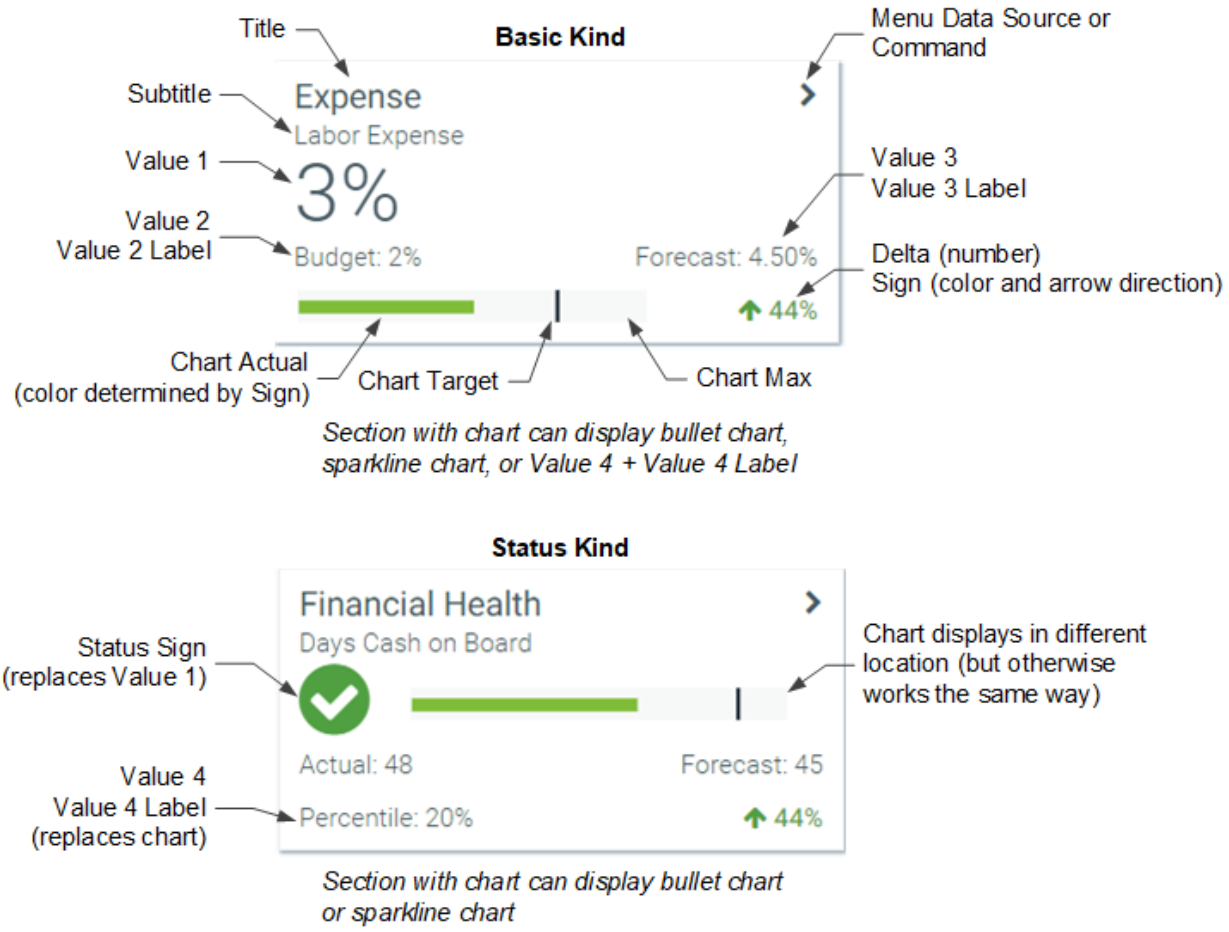
- **Use a KPI table in the Reports Library.** In this case, the component reads the KPI data directly from the table. The KPI data is never brought into the spreadsheet. This approach requires a KPI table to be present in the Reports Library, and that table must be populated with the desired KPI data using some other means. For example, you may have a separate report utility that runs periodically to calculate the KPI data and save it to the KPI table. For more information on KPI tables, see the *System Administration Guide*.

The KPI Panel component depends on a predefined data structure so that it can automatically format and position the KPI data into a series of KPI boxes. KPI tables and the KPISource data source both use the same basic predefined structure to provide the KPI data.

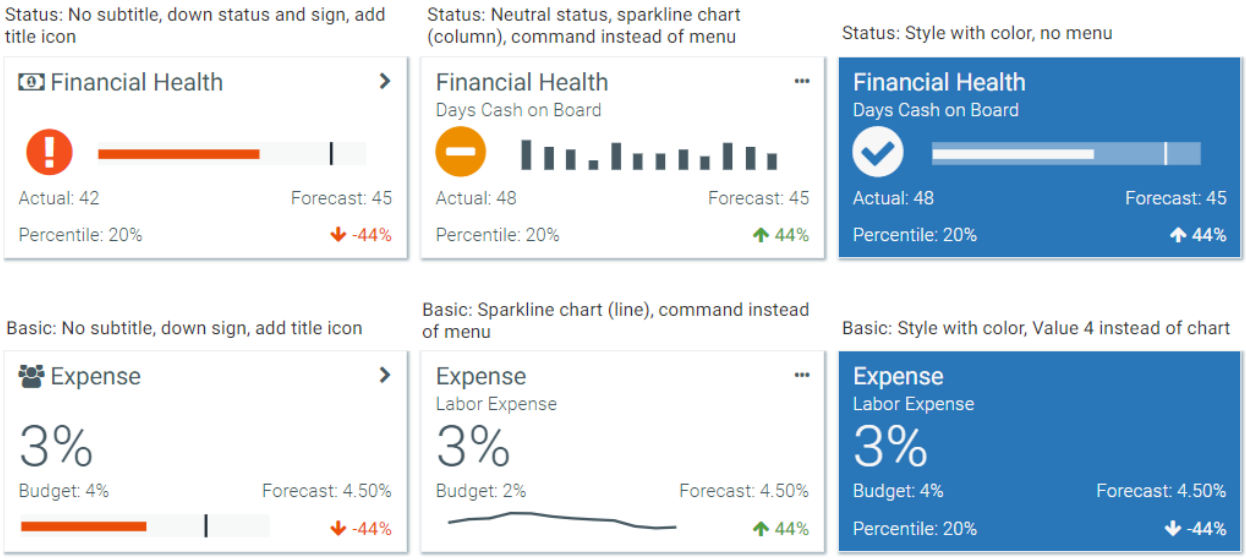
The KPI boxes in the panel can display in a variety of ways, depending on the following:

- **KPI Kind**: The kind specified for each KPI determines the information that displays in each KPI box. **Basic** KPIs emphasize numeric detail, whereas **Status** KPIs are intended to show whether a KPI is "good" or "bad" at a glance. Basic and Status KPIs can be mixed within the same panel.
- **Other KPI Properties**: The various properties that are populated for each KPI affect the display of that KPI. For example, a Basic KPI can display either Value 4 or a chart, but not both.
- **KPI Size**: The KPI size is set at the component level and determines how much detail displays in each box.

The following diagram shows how the major KPI properties are displayed in Basic and Status KPIs, so that you can see how the data structure maps to the presentation of KPI boxes.

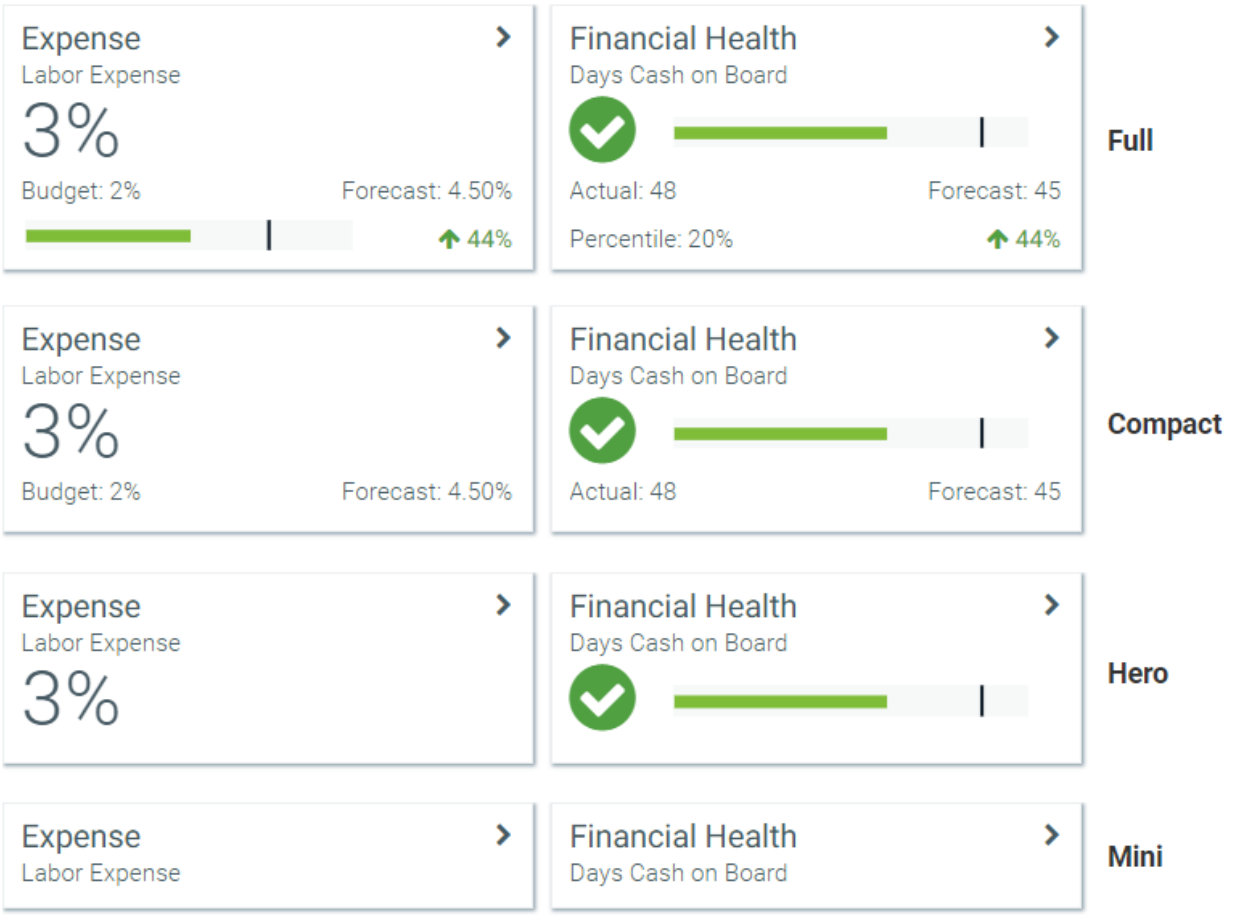


The following screenshot shows some common variations on this structure:



NOTE: If the value of any property is too long to display in its allotted space, the value is truncated and displays with an ellipsis. The full value is shown in a tooltip.

The following screenshot shows how the specified size affects the presentation of Basic and Status KPIs.

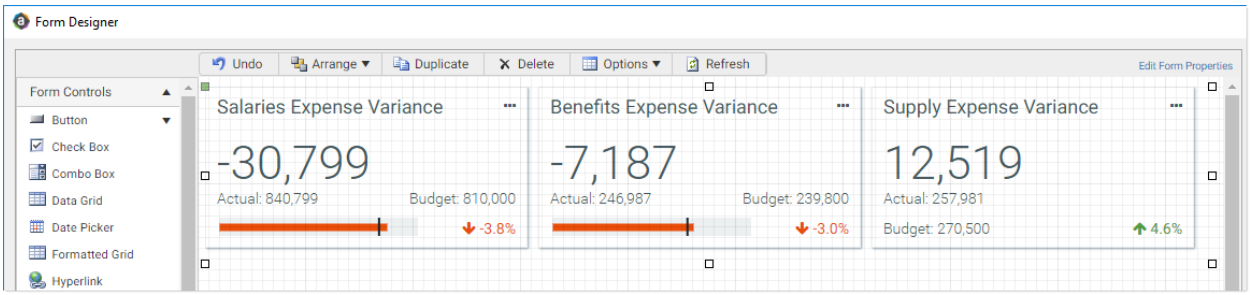


Sizing the KPI Panel on the canvas

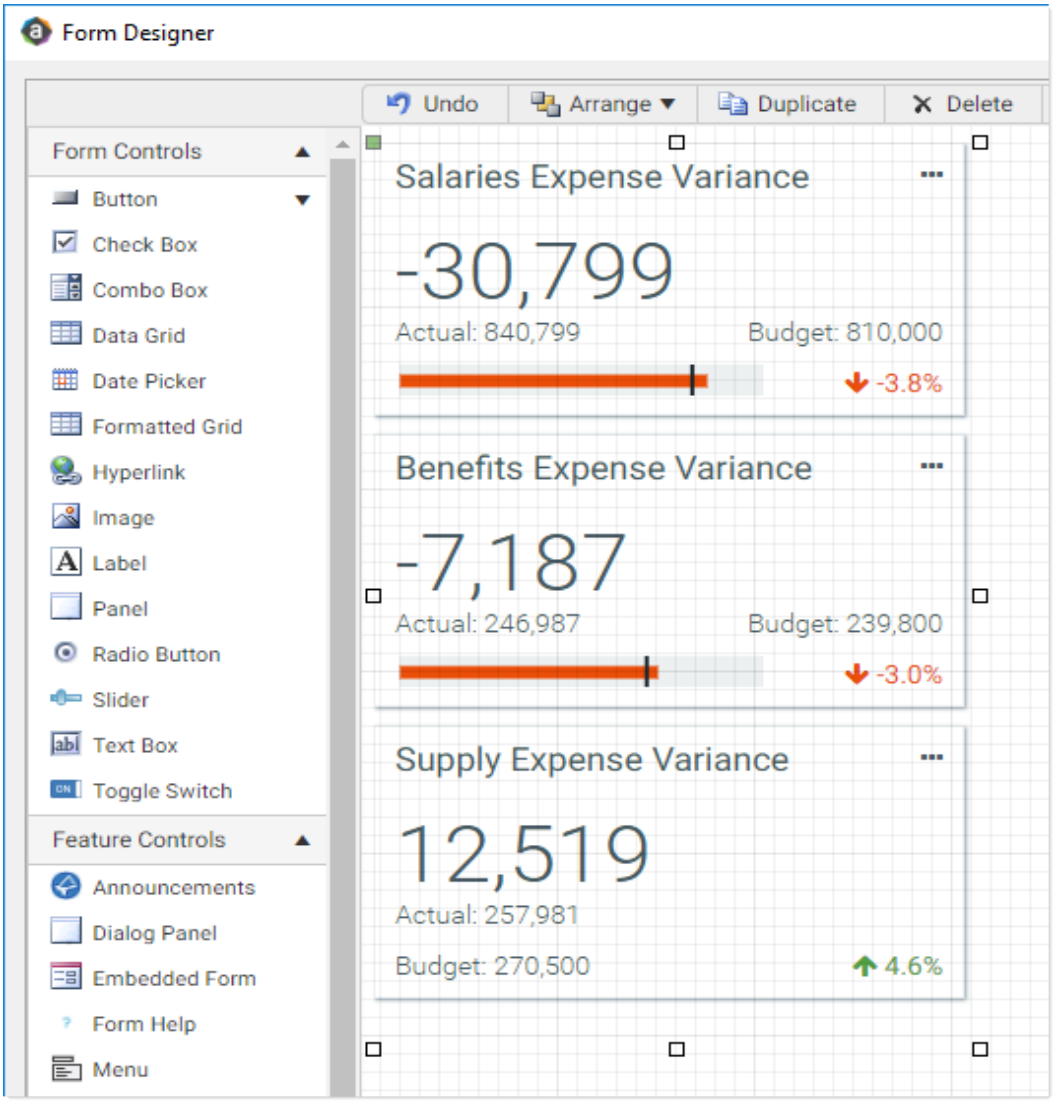
The KPI Panel is a specialized panel component that is designed to show one or more KPI "boxes". The number of KPIs to display within the component depends on the KPI rows defined in the KPISource data source or in the KPI table.

The KPI Panel uses panel flow behavior to automatically position the KPI boxes within the component. You should size the KPI Panel component depending on how many KPIs are defined in the data source and how you want those KPIs to be arranged in the form.

For example, imagine that you have 3 KPIs and you want those KPIs to display in a horizontal bar. In that case, you should size your KPI Panel to be long and short, so that the KPIs all fit on the same "row" and flow horizontally:



If you want the KPIs to display as a vertical bar down the side of the form, you should size your KPI Panel to be tall and thin, so that each KPI flows down to a new "row":



If you want the KPIs to stack, size the KPI panel as wide as necessary to fit the number of KPI boxes that you want to display horizontally, and as tall as necessary to fit the number of KPI boxes that you want to display vertically.

The exact component size necessary to fit the KPIs depend on the following:

- The **Child Padding X** and **Child Padding Y** settings, which set the padding between each KPI box.
- The **Size** setting, which determines the height of each KPI box.

For example, the Full size KPI is approximately 300px wide by 150px tall (plus the padding).

NOTE: The overflow behavior for the KPI Panel is set to Auto and cannot be changed. This means that if the number of KPI boxes exceeds the space allotted to the KPI Panel component, the component displays with a scroll bar to view the excess KPIs. The KPIs will not overflow the panel or be cut off as hidden.

Within the KPI Panel, the KPI boxes are ordered as follows:

- When using a data source, the boxes are displayed in the order they are found in the data source.
- When using a KPI table, you can specify a sort order for the boxes based on one or more columns in the table.

Data source tags

KPI Panel components can use a KPISource data source to define the KPIs to display in the component. The tags for the data source are as follows:

Primary tag

`[KPISource;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a KPI Panel component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[KPI]`

Each row flagged with this tag defines a KPI to display in the KPI Panel component. Each KPI row displays as a distinct KPI "box" in the form.

Column tags

[ID]

Optional. The ID for the KPI. This column can contain any value that uniquely identifies each KPI row. This column is only necessary if you want to implement interactivity for the form based on the currently selected KPI, or based on the KPI where a command was triggered using its menu. If you do not need this column, it can be omitted or left blank. For more information, see Interactive behavior and Executing commands from KPI Panels.

[Kind]

The kind of KPI, either `Basic` or `Status`. The kind determines whether the primary KPI value is numeric or a status icon. For more information about how each KPI kind displays, see the Component overview.

If this column is blank, or if it contains any value other than Basic or Status, the KPI kind is interpreted as `Legacy`. The Legacy kind is for backward-compatibility only, and it displays KPIs using the format and properties supported by KPIs created in 2018.1. For more information, see Legacy KPIs.

[Title]

The title of the KPI, displayed at the top of the KPI box.

[SubTitle]

Optional. The subtitle of the KPI, displayed directly beneath the title.

[TitleIcon]

Optional. The name of an icon to display in the KPI title. Enter any valid icon name, such as `fa-dollar`. The icon names are the same as the symbol names available for use in Axiom form components such as Formatted Grids. If specified, the symbol displays in the far left of the title, before the title text.

To look up valid icon names, you can use the symbol choosers available for Formatted Grid, Label, and Button components. Currently, no helpers are available to populate the TitleIcon column with icon names directly.

[Value1]

The primary value to highlight for the KPI, when using the Basic kind. This value displays in large, bold font directly underneath the title. This is the value that you want to draw the most attention to. The value displays using the number format defined for the cell.

If you are using the Status kind, this value is ignored and instead the primary value is the StatusSign.

[Value2]

Optional. A supporting value to show for the KPI. A label can be defined for this value, using the Value2Label column. The value displays using the number format defined for the cell.

The supporting values can be used to provide additional information about the primary value. For example, if the primary value is a variance, then Value 2 and Value 3 might display the actual and budget numbers used to calculate that variance. Or if the primary value is the actual number, then Value 2 and Value 3 might display the variance and the budget number to provide more context for the actual number. The values displayed are entirely user-definable.

[Value2Label]

Optional. The label for Value 2. The label precedes the value and displays with a colon, such as "Actuals: *Value 2*". The label should explain what Value 2 represents.

[Value3]

Optional. A supporting value to show for the KPI. A label can be defined for this value, using the Value3Label column. The value displays using the number format defined for the cell. See the description of Value2 for more information.

[Value3Label]

Optional. The label for Value 3. The label precedes the value and displays with a colon, such as "Budget: *Value 3*". The label should explain what Value 3 represents.

[Value4]

Optional. A supporting value to show for the KPI. A label can be defined for this value, using the Value4Label column. The value displays using the number format defined for the cell. See the description of Value2 for more information.

When using the Basic kind, Value 4 and the chart are interchangeable. You can display either Value 4 or a chart, but not both. If both are defined, the chart takes precedence. When using the Status kind, the chart displays in a different place so you can display both if desired.

[Value4Label]

Optional. The label for Value 4. The label precedes the value and displays with a colon, such as "Forecast: *Value 4*". The label should explain what Value 4 represents.

[ChartTarget]

Optional. A value that defines the target line for the bullet chart. This value can be omitted if it is not needed.

[ChartActual]

Optional. A value that defines the actual line for the bullet chart. This value can be omitted if the KPI does not use a bullet chart.

[ChartMax]

Optional. The maximum value of the bullet chart. The chart target and actual values are represented in relation to this maximum value.

For example, if the actual value is 100 and the maximum value is 1000, then the actual bar will only take up 1/10th of the bullet chart. But if the maximum value is 150, then the actual bar will take up 2/3rds of the bullet chart.

This value is required if you want to display a bullet chart on the KPI. This value should be omitted if you don't want to display a chart at all, or if you want to display a sparkline chart instead by using the SparklineDataSource column. If ChartMax and SparklineDataSource are both defined, an error occurs when attempting to render the component.

[Delta]

Optional. A value that illustrates the positive or negative measure of the KPI. This value can be omitted if not needed.

The Delta value displays in either red or green (as determined by the Sign value), using the number format defined for the cell. The Delta value can be used to show a variance percent or a raw difference value. It can also be used to show the change in value since the last time the primary KPI value was measured.

[Sign]

Optional. Specifies whether the primary KPI value is trending up (positive) or down (negative). Enter either `Up` or `Down`. If omitted, Down is assumed.

- If Up, then an up-arrow displays in front of the Delta value. The value, arrow, and the actual bar of the bullet chart display in green.
- If Down, then a down-arrow displays in front of the Delta value. The value, arrow, and the actual bar of the bullet chart display in red.

If the style of the KPI is anything other than `white`, `S1`, or blank (transparent), then these items display in white instead of green or red.

If the KPI does not have a defined Delta value, the Sign still determines the color of the bullet chart (if applicable).

[StatusSign]

Specifies the status of the KPI, when using the Status kind. Enter one of the following: `Up`, `Down`, `Neutral`. The status displays as a colored circle with a positive, neutral, or negative indicator:

The green, orange, and red colors are only used when the style of the KPI is `white`, `S1`, or blank (transparent). If the box has a background color, then the status circle is white and the indicator uses the same color as the background.

[Command]

Optional. Specifies a command to execute when the user clicks the icon in the top right corner of the KPI box.

If you want users to be able to execute a command from the KPI box, you can use the Command column or you can use the MenuDataSource column.

- When using Command, you can define a single command to be triggered by a three-dots icon that displays in the top right corner of the box. This option is intended for cases where you only need to provide access to one command, and you don't need a custom icon.
- When using MenuDataSource, you can define one or multiple commands in a separate KPIMenu data source. This option is intended for cases where you need to present multiple command options to the user, or if you need to specify a custom icon for a single command.

The valid entries for the Command column are the same that can be defined for the Command column in the KPIMenu data source. See Executing commands from KPI Panels.

[Style]

Optional. Specifies a color style to set the background color of the KPI box. By default, the box is transparent.

The following Axiom color styles are supported (specify one per KPI): S1, S6, A11, A51, P5, P6, P7, P9, P10. You can also specify `white`. When using darker background colors, the text in the KPI automatically adjusts to white.

[Tooltip]

Optional. Defines a tooltip to display when a user hovers over the button in the top right corner of the KPI box. This applies as follows:

- If you are using the Command column, the tooltip displays for the default three-dots icon.
- If you are using the MenuDataSource column and the KPIMenu data source contains multiple commands, the tooltip displays for the carat icon that opens the menu.
- If you are using the MenuDataSource column and the KPIMenu data source contains one visible command, the tooltip defined for that command displays on the custom icon (instead of this tooltip).

[MenuDataSource]

Optional. Specifies a KPIMenu data source that defines one or more commands that can be executed from the KPI box. If multiple commands are available, these commands display in a fly-out menu. For more information, see Executing commands from KPI Panels.

If both the Command column and the MenuDataSource column are populated, the menu takes precedence.

[SparklineDataSource]

Optional. Specifies a chart data source in order to display a sparkline chart in the KPI box. The data source must be an XYChart data source. For more information, see Displaying charts in KPI Panels.

The KPI box can display a bullet chart or a sparkline chart, but not both. If ChartMax and SparklineDataSource are both defined, an error occurs when attempting to render the component.

[SparklineSeriesName]

Optional. Specifies a series name within the data source specified as the SparklineDataSource. The series must be Line or Column kind within the XYChart data source. Unsupported kinds are rendered as Line.

Only one series can be displayed within the sparkline chart. If the series name is omitted or invalid, the sparkline chart does not display. For more information, see Displaying charts in KPI Panels.

[Hidden]

Specifies whether the KPI row is hidden from the KPI Panel (True/False). If omitted, False is assumed. You can use a formula in this property to dynamically show or hide the KPI based on some condition.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

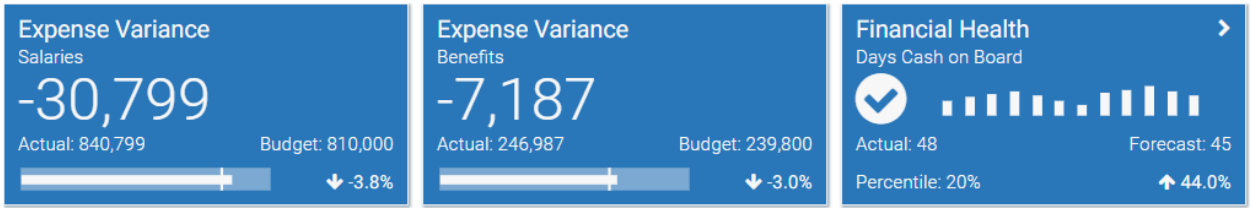
To use the Data Source Wizard to add the tags to a sheet, right-click in a cell and then select **Create Axiom Form Data Source > KPI Panel**. You can also highlight a range of data first and then use the wizard to add the tags around that data. The cells in the row above and the column to the left of the selected area must be blank in order for Axiom to place the tags in sheet.

The following example shows a sample KPISource data source tagged in a sheet:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
2																		
3			[KPISource;Expenses]	[Kind]	[Title]	[SubTitle]	[StatusSign]	[Value1]	[Value2]	[Value2Label]	[Value3]	[Value3Label]	[ChartTarget]	[ChartActual]	[ChartMax]	[Delta]	[Sign]	[Style]
4			[KPI]	Basic	Expense Variance	Salaries		-30,799	840,799	Actual	810,000	Budget	810,000	840,799	1,000,000	-3.8%	Down	P7
5			[KPI]	Basic	Expense Variance	Benefits		-7,187	246,987	Actual	239,800	Budget	239,800	246,987	350,000	-3.0%	Down	P7
6			[KPI]	Status	Financial Health	Days Cash on Board	Up		48	Actual	45	Forecast				44.0%	Up	P7

Example KPISource data source

The resulting KPI Panel for the example data source shown above looks as follows:



As discussed in the previous section, the arrangement of multiple KPI boxes depends on the size of the KPI Panel component on the canvas. In this example, the KPI Panel has been sized long and short to display the three KPIs horizontally.

Once the KPISource data source has been added to the sheet, you can optionally use the Data Source Assistant to:

- Add column and row tags to the data source.
- Review and complete column properties for each row in the data source. When your cursor is in a `[KPI]` row, all available column properties display in the Selection Editor. You can edit any property and that edit will be made in the corresponding cell of the data source.

Component properties

You can define the following properties for a KPI Panel component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source to provide the KPI data for the component. You must have defined the KPISource data source within the file using the appropriate tags in order to select it for the grid. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file. **NOTE:** KPI Panel components can use either a data source or a KPI Table. If you specify a data source, the table-related properties are hidden in the Form Assistant and Form Designer. If you later decide you want to switch from using a data source to a table, you must clear the data source name in order to show the table-related properties.
Enable KPI Selection	Optional. Specifies whether users can select a KPI box in the panel. Only applies when using a data source. By default this is disabled, which means KPIs are not selectable in the panel. If enabled, and if the ID column is populated in the KPISource data source, then KPIs are selectable in the panel. When a user selects a KPI, the ID for that KPI is written back to **Selected ID** field. The form can be configured to change in some way based on the currently selected KPI. For more information, see Interactive behavior.

Item	Description
Selected ID	Optional. The currently selected KPI in the panel, identified by the corresponding ID in the KPISource data source. Only applies when using a data source, and only if **Enable KPI Selection** is enabled. This setting serves two purposes: • It specifies the initially selected KPI in the panel, when the user first opens the form. You can leave this setting blank to have no initial selection, or you can enter an ID from the data source into the Selected ID field. • When a user views the form and selects a KPI in the panel, the ID of the selected KPI will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected KPI in the panel. For more information, see Interactive behavior. **NOTE:** This component always auto-submits when a user selects a KPI in the panel.
Triggering ID	When a user executes a command from a KPI box, the ID for that KPI is written back to this field (as defined in the KPISource data source). You can reference this value to impact the command being executed, or to impact something else in the form. For more information, see Using the Triggering ID to impact the form. **NOTE:** This setting is only available on the Form Control Sheet; it does not show in the Form Assistant or in the Form Designer.
KPI Table	The KPI table to provide the KPI data for the component. Enter any KPI table name. Only KPI tables can be used in this context, because KPI tables contain the necessary columns that map to the properties used by the KPI Panel component. For more information on KPI tables, see the *System Administration Guide*. **NOTE:** KPI Panel components can use either a data source or a KPI Table. If you specify a KPI table, the data source-related properties are hidden in the Form Assistant and Form Designer. If you later decide you want to switch from using a table to a data source, you must clear the table name in order to show the data source-related properties.

Item	Description
Data Filter	Optional. A filter to limit the KPIs shown in the component. Enter any valid filter criteria statement based on the specified KPI table. Only applies when using a KPI table. If no filter is defined, then all KPIs in the table will display in the component (except for rows with the **Hidden** column set to **True**).
Sort Order	Optional. One or more table columns to determine the sort order of the KPIs shown in the component. Specify any columns on the KPI table or a lookup table. Only applies when using a KPI table. Use fully qualified Table.Column syntax to specify each column. Separate multiple columns with semicolons. If multiple columns are specified, the first column is the top-level sort. You can optionally indicate ascending (asc) or descending (desc) sort after the column name. Ascending order is used by default. For example: `Dept.Region desc; Dept.Dept`.
KPI Size	Specifies the size of the KPI boxes in the panel. KPIs support the following sizes: • **Full** (default): All available KPI values are shown, using the full size of the box. • **Compact**: The bottom row of values does not display in the KPI. This includes Value 4 and its label, the delta value and the sign indicator, and the chart for Basic KPIs. This is intended for KPIs where you have some supporting values but you do not need the full level of detail. • **Hero**: Only the most important values display in the KPI, including the title and subtitle, menu or command, Value 1 for Basic KPIs, and the status sign and chart for Status KPIs. This is intended for KPIs where you do not need to show any supporting values. You only want to communicate the primary value or status. • **Mini**: Only the title, subtitle, and menu or command display in the KPI. This is intended for cases where the KPI Panel component is being used solely as a selector tool or to execute actions. For examples of each size, see the screenshot of sizes in the Component overview.
Child Padding X Child Padding Y	Defines the x-padding and y-padding between KPI boxes. • Child Padding X defines the horizontal padding between KPI boxes. It is applied to the right side of each box. • Child Padding Y defines the vertical padding between KPI boxes. It is applied to the bottom of each box. The padding can be set in pixels (default) or in percentages.

Item	Description
Component Dependencies	Optional. Specifies one or more components that the KPI Panel component is dependent on. Only applies when using a KPI table. If you want the KPI panel to dynamically update based on changes made to other components, list one or more component names in this field. Separate multiple component names with commas. If a component name is listed here, then the KPI panel is refreshed when a form update submits a change to the listed component. If no component names are listed here, or if the listed components are unchanged, then the KPI panel is not refreshed when a form update occurs. Components listed as component dependencies must be interactive components, such as Combo Box components, Check Box components, and so on. The purpose of this option is that you want to enable refreshing the KPI panel based on a change a user made to an interactive component. Non-interactive components, such as Label components, cannot submit values back to the source file and cannot trigger form updates. Therefore, non-interactive components cannot cause the KPI panel to refresh. **NOTE:** Standard Button components can be used as component dependencies. If a button uses the default Command behavior, then whenever the listed button triggers a form update, the KPI panel will be refreshed. However, if the button uses a specialized button behavior, or if the button uses a command that alters the normal form update behavior, then the button may not cause the KPI panel to refresh. For more information, see Update behavior.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

By default, the KPI boxes in the panel display with a border and with a shadow. You can apply styles to the component in order to remove either of these items:

- **no-kpi-box-shadow**: Removes the shadow on individual KPI boxes.
- **no-kpi-border**: Removes the border on individual KPI boxes.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Update behavior

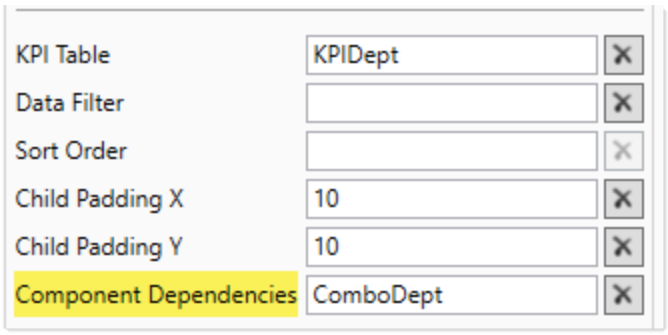
The KPI Panel component uses two different types of update behavior, depending on the source of data for the KPIs:

- **Data source**: When using a data source, the update behavior is the standard form update behavior. Each time a form update is triggered, the KPI Panel component is refreshed to reflect the current component settings and show the latest data in the data source.
- **KPI table:** When using a KPI table, the KPI Panel component queries data from the Axiom Software database when the Axiom form is initially rendered. This data remains the same until one of the following occurs:
    - If the form uses refresh variables, applying changed refresh variables via the Filters panel will refresh the KPI panel. This means that the KPI panel can be set up to change its data based on the selected value of a refresh variable.
    - If one or more components are listed in the **Component Dependencies** property for the KPI panel, the KPI panel is refreshed when a changed value is submitted for one of those components. Otherwise, if no components are listed, or if no changes are submitted for listed components, then form updates triggered by interactive components do *not* cause the KPI panel to refresh.

By default, when an update is triggered in the form, the KPI panel is preserved as is. The component settings are not re-read and the data query is not run again. This behavior is intended to improve performance by not executing the data query and not redrawing the panel every time a form update occurs.

For example, imagine that the form contains a Combo Box component that is set to auto-submit. When a user selects a value from the combo box, this value is submitted to the source file and a form update is triggered. Under normal circumstances, if another component is configured to dynamically change based on the currently selected value for the combo box, this change would be reflected in the form once the form update is complete. However, the KPI Panel component does *not* update in this circumstance. Even if the selected value for the combo box impacts a KPI Panel property—such as the KPI table or the data filter—by default the KPI panel will not change during this form update.

If you want the KPI panel to update based on the selected value of the combo box, then you must list the name of the Combo Box component in the **Component Dependencies** property for the KPI Panel component. For example, if the Combo Box component is named ComboDept because it is used to select a department, you would list ComboDept as a component dependency.



Now when a change is submitted for the Combo Box component named ComboDept, the KPI Panel component is refreshed. The data query is run based on the current component properties. This occurs at the end of the form update process, when the form display is updated in the browser.

When a form update is triggered, Axiom Software checks to see if any component names are listed in the **Component Dependencies** property of the KPI Panel component. You can list multiple component names, separated by commas. If any components are listed, Axiom Software then checks to see if any changes to those components are included in the current form submission. If the listed components are unchanged, the KPI Panel component is not refreshed during the form update. If one or more of the listed components are changed, then the KPI Panel component is refreshed.

NOTE: The components in Component Dependencies do not have be set to auto-submit in order to refresh the KPI Panel component. If an interactive component is changed but it is not configured to auto-submit, then its change will be submitted when the next form update is triggered (either by a Button component, or by a different component that is configured to auto-submit). The KPI Panel component will still recognize the component change, even though the change was submitted by a different component.

Interactive behavior

The KPI Panel component can be set up to allow the user to select a specific KPI box within the panel, and then submit the selected KPI back to the source file. The selected KPI is written to the **Selected ID** setting on the Form Control Sheet, using the ID value defined in the KPISource data source.

To enable selecting rows:

- **Enable KPI Selection** must be enabled for the KPI Panel component.
- The KPISource data source must contain a populated ID column. If the ID column is omitted or blank, then row selection is not available. Row selection is also not supported when using a KPI table.

If you want the Axiom form to respond to the currently selected KPI, then you must set up the file so that another component references the selected KPI and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

KPI interactivity is intended to support KPI "drilling" based on the currently selected item. For example the user may want to see more detail about the data that makes up the KPI, or see "child" KPIs related to the selected KPI.

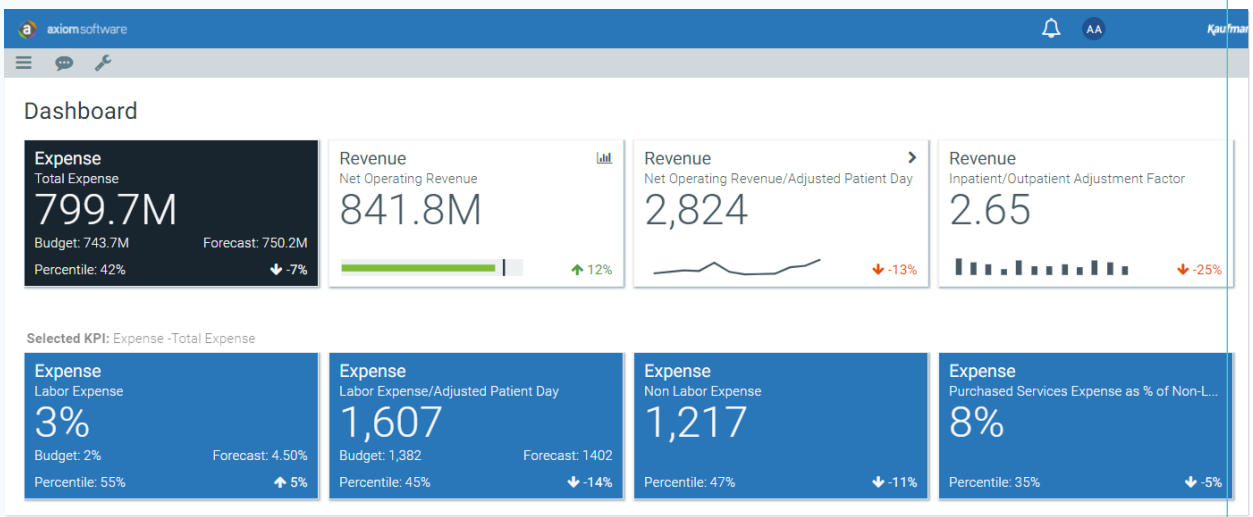
When a user selects a KPI in the panel, the selected KPI box updates to display with a black background color. This is how the user knows which KPI is selected. Black is not otherwise supported as a background color for KPI boxes, so it does not conflict with any other background color that may be used in the panel.

Example

The Axiom form could contain an initial KPI Panel component that shows top-level key performance indicators. If you want users to be able to see lower-level KPIs within the same category, you could set up a secondary KPI panel that shows these "child" KPIs based on the currently selected KPI in the top panel. The secondary panel could remain hidden until a KPI is selected, or the form could start out with a default selection.

For example, the initial KPI Panel could have top-level KPIs for Revenue and Expense. When the user clicks the KPI box for Expense, the secondary panel could show additional detailed KPIs relating to expenses. The secondary panel could be set up in a variety of ways in order to change what it shows:

- The form could contain multiple data sources to be used by the secondary panel, and the selected KPI could drive which data source the secondary panel uses.
- The form could contain a single data source for the secondary panel, and the selected KPI could drive which KPIs in the data source are currently visible (using the `[Hidden]` column).
- The secondary panel could reference a KPI table, and the selected KPI could drive which table is referenced, or drive a filter on the table.



Example form with selected KPI

Legacy KPIs

Legacy KPIs refer to the KPI structure in version 2018.1, before additional kinds and sizes were introduced. If you have KPI Panels created in version 2018.1, these KPIs are displayed as legacy KPIs until you assign them a kind (and complete any additional properties used by the kind).

Legacy KPIs are intended for backward-compatibility only. Going forward, all new KPIs should be assigned a kind.

Legacy KPIs are most similar to the Basic kind, but have the following differences:

- The formatting applied to legacy KPIs has minor, subtle differences. For example, the spacing between the top of the box, the title, and Value 1 is slightly different. If a legacy KPI is displayed next to a Basic KPI, they will not look exactly the same, though they use the same basic elements.
- Legacy KPIs do not support subtitles, Value 4, status signs, or sparkline charts. The only chart option is the bullet chart.
- Legacy KPIs do not support sizes. Legacy KPIs always display at full size, regardless of the size set in the KPI Panel component properties.
- If no bullet chart is defined for a legacy KPI, the display of the values rearranges as follows:

Executing commands from KPI Panels

You can configure a KPI box in a KPI Panel component to execute one or more commands. For example, you may want to launch a file with supporting information about the KPI, or open a Dialog Panel to show supporting information.

There are several different ways that you can configure commands for KPIs. The approach to use depends on whether you need to execute one command or multiple, and whether you need to use custom icons with the command.

- **Basic Single Command:** If you only need to execute a single command, and you don't need a custom icon, then you can define the command in the Command column (either in the KPISource data source or in a KPI table). The KPI box displays with a three-dots icon in the upper right corner. Users can click this icon to execute the command.

- **Custom Single Command:** If you want to display a custom icon with a single command, then you can use a separate KPIMenu data source to define the command and its icon. The KPI box displays with the custom icon in the upper right corner. Users can click this icon to execute the command.
- **Custom Menu with Multiple Commands**: If you need to present multiple command options to users, then you can use a separate KPIMenu data source to define these commands. The KPI box displays with a carat icon in the upper right corner. Users can click this icon to open a fly-out menu that displays all of the commands using their defined names and icons.

Example KPIs with commands

The KPIMenu data source is associated with the KPI in different ways, depending on whether you are using a KPISource data source or a KPI table.

- If you are using a KPISource data source, then the KPIMenu data source name is placed in the MenuDataSource column.
- If you are using a KPI table, then you can save the contents of the KPIMenu data source to the MenuData column using Save Type 1. This must be part of the save-to-database file that is being used to save data to the table. See Saving KPIMenu values when using a KPI table for more information.

Valid command strings for use in KPIs

Command strings for KPIs can be any of the following items:

Valid Commands	Description
URL	Specify a URL (starting with HTTP/S) to open a web page, Axiom form, or web report. For example, you can use GetFormDocumentURL or GetWebReportDocumentURL to generate a URL to another Axiom file and launch it from the KPI.
Document shortcut	Specify a document shortcut to a file in the Axiom Software file system. Document shortcuts use the syntax `document://filepath`. For example: `document://\Axiom\Reports Library\Reports\expense_analysis.xlsx`

Valid Commands	Description
Command	Specify a command string to execute a command from the Command Library. For example: `command://ShowFormDialogPanel?DialogPanel=DialogPanel1` Multiple commands can be combined into a single command string, separated by commas. The command strings use the same syntax supported by the Button tag for Formatted Grid components. If the command string is invalid, no error displays and no action occurs when a user clicks on the menu item.

To use a command from the Command Library, right-click the cell and select **Axiom Wizards > Command Wizard**. This opens the **Shortcut Properties** dialog. Click the **[...]** button to the right of the **Shortcut Target** box to open Axiom Explorer, then navigate to the Command Library to select a command. You can then configure the shortcut properties for the selected command. When you click **OK**, the command string is inserted into the cell. You can use any command that is supported for use in Axiom forms, though some commands may not make sense to execute from a KPI Panel. (The Command Wizard can also be used to create a document shortcut, by selecting a document as the Shortcut Target.)

When a command from the Command Library is used, the form update process is triggered (if supported by the command). The behavior is essentially the same as when a regular Button component is used to execute the command. If the command is a URL or a document shortcut, the form update process does not occur.

Creating a KPIMenu data source

Using the KPIMenu data source, you can define one or more commands to display with custom icons on a KPI box. When using the data source with a KPI table, the data source must be created in the save-to-database file that is being used to populate the table, instead of in the form source file.

The tags for the data source are as follows:

Primary tag

[KPIMenu;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a KPI in a KPI Panel. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[MenuItem]

Each row flagged with this tag defines an item to display in the menu.

Column tags

[ID]

An ID that uniquely identifies each row in the data source. The ID can consist of numbers, text, or a combination of both, as long as it is unique for each row.

[Name]

The name of the menu item. This is the text that displays on the menu. The user clicks on the text to execute the menu item.

[Icon]

The name of an icon to display in the menu for this menu item. Enter any valid icon name, such as `fa-bar-chart`. The icon names are the same as the symbol names available for use in Axiom form components such as Formatted Grids.

To look up valid icon names, you can use the symbol choosers available for Formatted Grid, Label, and Button components. Currently, no helpers are available to populate the Icon column with icon names directly.

[Tooltip]

Optional. Defines text to display in a tooltip when a user hovers their cursor over the menu item.

[Command]

The command to execute when a user clicks the menu item. For more information, see Valid command strings for use in KPIs.

[Disabled]

Optional. Specifies whether the item is disabled on the menu (True/False). The default value is False if omitted or blank.

If True, then the item continues to display on the menu, but it is grayed out and cannot be selected. This option can be used to dynamically enable or disable a menu item based on a condition.

[Hidden]

Optional. Specifies whether the item displays on the menu (True/False). The default value is False if omitted or blank.

If True, then the item does not display on the menu. This option can be used to dynamically show or hide a menu item based on a condition.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

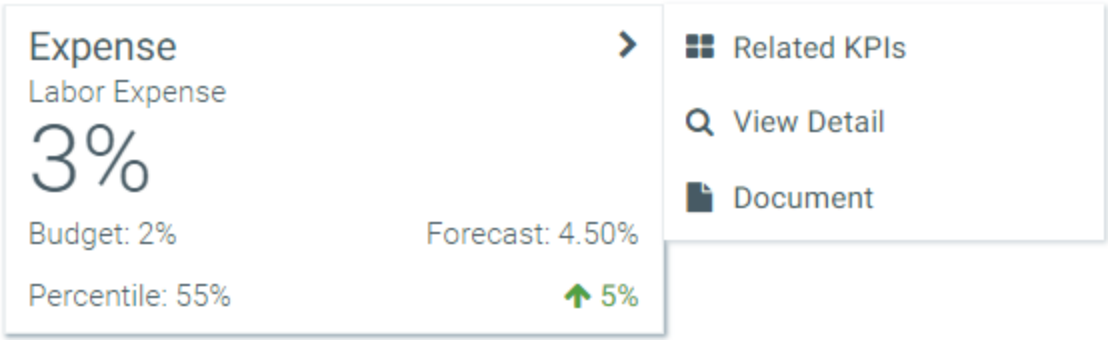
The following example data source defines a KPI menu with three items:

	B	C	D	E	F	G	H
14							
15		[KPIMenu;KPIMenuSourceMultipleItems]	[ID]	[Name]	[Icon]	[Tooltip]	[Command]
16		[MenuItem]	1	Related KPIs	fa-th-large	Related KPIs	command://ShowFormDialogPanel?DialogPanel=DialogPanelDetailKPI
17		[MenuItem]	2	View Detail	fa-search	View Detail	command://ApplySharedVariables?ShowChart=On
18		[MenuItem]	3	Document	fa-file	File	document://\Axiom\SystemFolderName_ReportsLibrary\Forms\report.xlsx?ds

Example KPIMenu data source

To use the Data Source Wizard to add the tags, right-click a cell and select **Create Axiom Form Data Source > KPI Menu**. You can right-click a single empty cell to place the initial tags and then fill out the data, or you can have the data already in the spreadsheet and highlight the applicable data to add the tags. The cells in the row above the data and the column to the left of the data must be blank in order for Axiom to place the tags in sheet.

The resulting menu would display on the KPI as follows:



Example fly-out menu on KPI

If the KPIMenu data source only contains one visible item, then that item displays directly in the top right corner of the KPI box, using the specified icon.

Saving KPIMenu values when using a KPI table

You can use a KPIMenu data source when saving KPI values to a KPI table. To do this, the save-to-database file that you use to save KPI data to the table must be set up as follows:

- The file must contain a KPIMenu data source. This data source is set up as normal, on any sheet of the file.
- When setting up Save Type 1 in the file, the contents of the MenuData column must contain the following special syntax to specify the KPIMenu data source to save:
  `[Datasource=`*`DataSourceName`*`]`.

For example, imagine that you have a KPIMenu data source named Menu, and you want to associate that data source with a KPI titled Expense. In the data to be saved to the database, the MenuData column for that KPI must contain the text `[Datasource=Menu]`.

	A	B	C	D	E
1					
2		[Save2db;DeptKPI;;;;;False]	Name	Title	MenuData
3					
4		[save]	Basic1	Expense	[datasource=menu]

Example save-to-database using special syntax to save KPI menu data

When the save-to-database is executed, Axiom Software finds the designated KPIMenu data source, and converts the contents of it into an XML string. That XML string is then saved to the MenuData column in the KPI table. When the KPI table is used with a KPI Panel component, the XML string is used to render the menu on the KPI box. The menu looks and acts the same way as when referencing the KPIMenu data source directly in a KPISource data source.

The MenuData column in KPI tables can only accept the special data source syntax when saving to the database using Save Type 1. If any other contents are present in the MenuData column within the sheet (even the resulting XML syntax), an error occurs when saving. If you want to modify and save the other

columns in the table without modifying the MenuData column, then the MenuData column must be omitted from the save.

Using the Triggering ID to impact the form

When a user clicks on a menu item to execute a command, the ID of the current KPI is written back to the **Triggering ID** field of the KPI Panel component. You can reference this value to impact the command being executed, or to impact something else in the form.

The Triggering ID only applies when using a KPISource data source in an Axiom form. It is not available when using a KPI table. The ID that is written to the Triggering ID field is from the ID column of the KPISource data source. The Triggering ID field does not display in the Form Assistant or Form Designer; it is only present in the Form Control Sheet.

For example, you may have a menu item that launches a Dialog Panel component, and you want to display the name of the current KPI in the dialog panel. The dialog panel can contain a Label component that uses a formula to reference the Triggering ID field. When the user clicks on the menu item, the Triggering ID is submitted back to the source spreadsheet (along with any other changed values in the form), and then displayed in the dialog panel.

The Triggering ID is submitted back to the form whenever a menu item is used to execute a command from the Command Library, assuming that the command triggers the regular form update process (some commands do not do this). This behavior also applies if the command is defined in the Command column instead of using a KPIMenu data source. If the menu item launches a URL or a document shortcut, the Triggering ID does not apply.

Displaying charts in KPI Panels

Each KPI in a KPI Panel component can include an optional chart. There are two options to display a chart:

- **Bullet Chart:** To display a bullet chart in the KPI box, complete the **ChartTarget**, **ChartActual**, and **ChartMax** columns in the KPI table or the KPISource data source.
- **SparklineChart:** To display a sparkline chart in the KPI box, first create an XYChart data source to define the data for the sparkline. Then, complete the **SparklineDataSource** and **SparklineSeriesName** columns in the KPISource data source, or the **SparklineData** column in the KPI table.

Each KPI can use either a bullet chart or a sparkline chart, but not both. If both ChartMax and SparklineDataSource are completed in a KPISource data source, an error occurs when rendering the component. If both ChartMax and SparklineData are completed in a KPI table, the bullet chart takes precedence.

Both kinds of KPIs (Basic and Status) can display charts. In Basic KPIs, the chart displays at the bottom left of the KPI box. In Status KPIs, the chart displays in the middle of the KPI box, next to the status indicator.

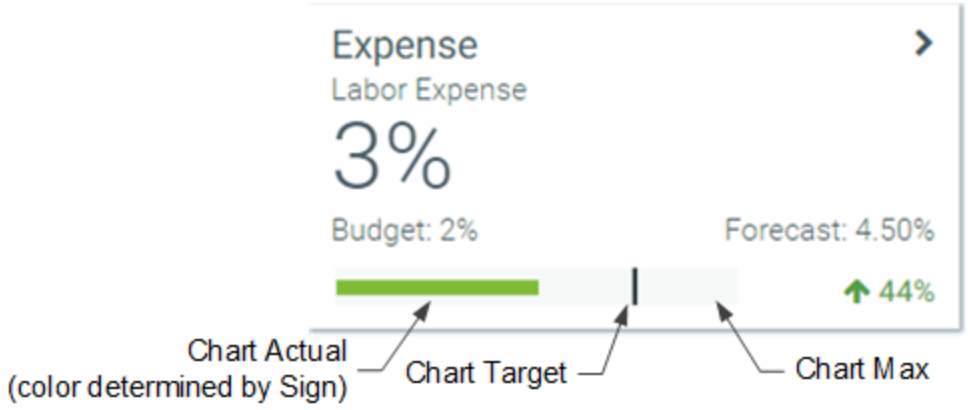
In Basic KPIs, the chart and Value 4 display in the same location, so only one or the other can be used per KPI. If both are defined, the chart takes precedence. This is not an issue for Status KPIs because the chart displays in a different location.

Displaying bullet charts in KPIs

To display a bullet chart in a KPI, complete the following columns in the KPI table or the KPISource data source.

- **ChartTarget**: A value that defines the target line for the bullet chart.
- **ChartActual**: A value that defines the actual line for the bullet chart.
- **ChartMax**: The maximum value of the bullet chart. The chart target and actual values are represented in relation to this maximum value.

  For example, if the actual value is 100 and the maximum value is 1000, then the actual bar will only take up 1/10 of the bullet chart. But if the maximum value is 150, then the actual bar will take up 2/3 of the bullet chart.



If the Style of the KPI is set to `white`, blank (transparent), or `S1`, then the actual bar of the chart displays in red or green, depending on the value of the Sign column. Otherwise, it displays in white.

Displaying sparkline charts in KPIs

To display a sparkline chart in a KPI, create an XYChart data source to define the sparkline data (see Creating an XYChart data source for a Sparkline chart). Then, do one of the following:

- If using a KPISource data source, enter the name of the XYChart data source in the **SparklineDataSource** column, and enter the name of a series in that data source in the **SparklineSeriesName** column. For example, the data source name might be `Sparkline` and the series name might be `Revenue`. The Revenue series is then displayed in the KPI box as a sparkline chart.

- If using a KPI table, you must use special syntax to save the contents of the XYChart data source and series to the SparklineData column of the table during a save-to-database. When setting up Save Type 1 in the file, the contents of the SparklineData column must contain the following special syntax: `[Datasource=DataSourceName;Series=SeriesName]`. The specified XYChart data source and series must be present in the same file where you are performing the save-to-database.

Sparkline charts do not display using colors. They are black when the KPI box uses a light color and white when the KPI box uses a dark color.

For example, you can create an XYChart data source named Sparkline, with two series. One is a line series named LineSeries and the other is a column series named ColumnSeries.

	AE	AC	AD	AE	AF	AG	AH
5							
6		[XYChart;Sparkline]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]
7		[XValueName]			Jan	Feb	March
8		[Series]	Line	LineSeries	5	10	15
9		[Series]	Column	ColumnSeries	1000	800	700

Example XYChart data source

When using a KPISource data source, you can list the series name and the XYChart data source name in the KPISource data source directly. When the KPIs are rendered in the form, Axiom Software finds the specified series in the specified XYChart data source, and displays the data as a sparkline chart.

	A	B	C	D	E	F	G	H
5								
6		[KPISource;Examples]	[ID]	[Kind]	[Title]	[SubTitle]	[SparklineDataSource]	[SparklineSeriesName]
7		[KPI]	Basic1	Basic	Expense	Labor Expense	Sparkline	LineSeries
8		[KPI]	Status1	Status	Financial Health	Days Cash on Board	Sparkline	ColumnSeries

Example KPISource data source referencing sparkline series

To save series data to a KPI table, you must place the special syntax in the SparklineData column for the save-to-database, as shown in the following screenshot:

	A	B	C	D	E
1					
2		[Save2db;DeptKPI;;;;;False]	Name	Title	SparklineData
3					
4		[save]	Basic1	Expense	[datasource=sparkline;series=lineseries]
5		[save]	Status1	Financial Health	[datasource=sparkline;series=columnseries]

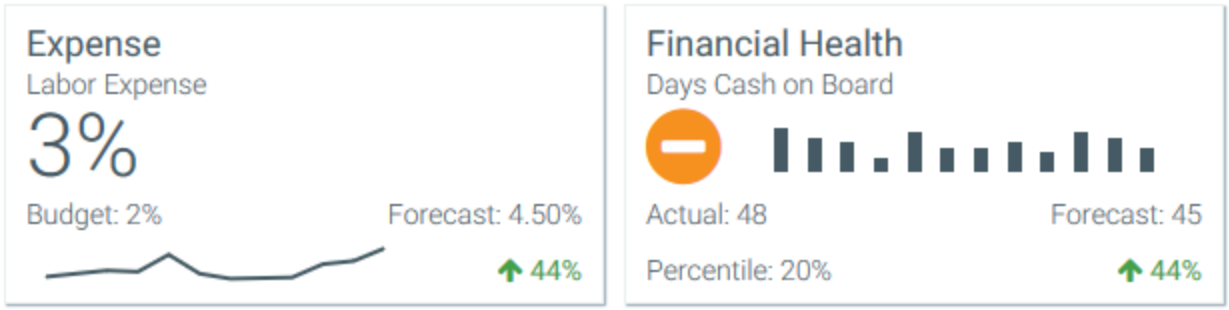
Example save-to-database using special syntax to save sparkline data

When the save-to-database is executed, Axiom Software finds the designated XYChart data source, and converts the contents of the specified series into an XML string. That XML string is then saved to the

SparklineData column in the KPI table. When the KPI table is used with a KPI Panel component, the XML string is used to render the sparkline chart on the KPI box. The sparkline chart looks and acts the same way as when referencing the XYChart data source and series directly in a KPISource data source.

The SparklineData column in KPI tables can only accept the special data source syntax when saving to the database using Save Type 1. If any other contents are present in the SparklineData column within the sheet (even the resulting XML syntax), an error occurs when saving. If you want to modify and save the other columns in the table without modifying the SparklineData column, then the SparklineData column must be omitted from the save.

The following example KPIs show how a line and column sparkline chart appear in the KPI box:



Example KPIs with sparkline charts

Creating an XYChart data source for a Sparkline chart

The tags for the XYChart data source are as follows when using it to define a sparkline chart for a KPI. When using the data source with a KPI table, the data source must be created in the save-to-database file that is being used to populate the table, instead of in the form source file.

Primary tag

`[XYChart;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a KPI. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each sparkline chart uses a single series in the data source.

Column tags

[SeriesName]

Defines the name of each series in the chart. The name identifies this series so that it can be assigned to a KPI.

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Line` or `Column`. Any other XYChart series kind listed here will render as Line.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

When using **Create Axiom Form Data Source** on the right-click menu, there is no separate option for Sparkline. Instead, you should select Line Chart or Column Chart to create an XYChart data source. You can modify the Kind column as needed to specify Line or Column for each series.

Line Chart component

Line Chart components display information as a series of data points connected to form a line. Line charts are part of the XYChart family, which includes bar, column, and area charts. All of these charts use the same data source type (XYChart) and have the same basic component properties.

Defining a line chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Line Chart component on the Axiom form canvas.

Line charts can also support form interactivity, to change the contents of the Axiom form based on the currently selected data point in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Line Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each series will use a different color.

[XValueName]

This row contains the names of each XValue column in the chart. These names will display along the primary axis of the chart (the X-axis for most charts; the Y-axis for bar charts).

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. These names will be displayed in the chart legend, if the chart is configured to show a legend (as defined in the component settings).

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`. If omitted, then all series in the chart will use the Default Series Kind as defined in the component settings. If a data source contains multiple kinds of series then it is known as a combination chart (for example, one or more column series combined with a line series).

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[Axis]

Optional. Specifies the Y-axis scale for each series. This column is only required if the chart has both a primary and secondary Y-axis. If omitted, the primary Y-axis scale is assumed. See Using two Y-axis scales with combination XYCharts.

[VisibleinLegend]

Optional. Specifies whether a particular series is shown in the chart legend (True/False). If omitted, all series are shown. This setting only applies if the chart is configured to show a legend (as defined in the component settings).

[ShowMarkers]

Optional. Specifies whether markers are shown on the line to indicate each specific data point in the series (True/False). If omitted, markers are shown. The marker is a circle with no fill color; the marker shape and fill are not configurable.

[LineStyle]

Optional. Specifies the style of the line as one of the following. If omitted, the Normal style is used.

- **None**: No line is displayed; only markers are shown to represent the data points. `[ShowMarkers]` must be enabled or else the series will not display at all. This option is primarily intended for use in combination charts—for example, multiple bar series combined with a marker-only line series.
- **Normal**: A straight line is drawn from point to point.
- **Smooth**: A curved line is drawn from point to point.
- **Step**: The line "steps" from one point to another. The lines between points are flat, with a vertical line up or down to indicate the differential at each point.

[DashType]

Optional. Specifies the type of dash as one of the following. If omitted, the Solid style is used.

- **Dash**: The line is drawn in dashes. The length of the dash is fixed and cannot be configured.
- **DashDot**: The line is drawn as a dash-dot-dash repeating series.
- **Dot**: The line is drawn in dots. The size of the dot is fixed and cannot be configured.
- **Solid**: The line is drawn as a solid line.

[PlotNullValues]

Optional. Specifies whether null values are plotted on the line (True/False).

If omitted or False, then null values will result in a gap between line segments. If True, then the missing value will be interpolated, so that the line will continue from the last plotted point to the next plotted point.

NOTES:

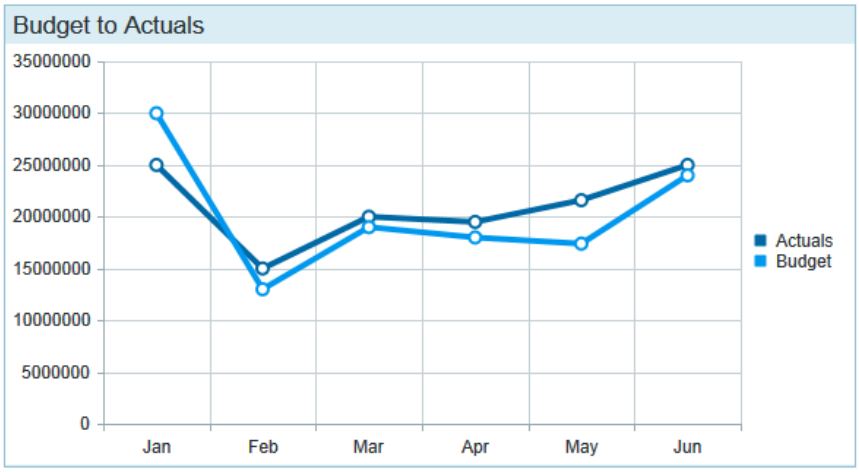
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple actual-to-budget data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J
1										
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Line	Actuals	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Line	Budget	30000000	13000000	19000000	18000000	17400000	24000000
6										

To use the Data Source Wizard to add the tags to a sheet, right-click a cell and then select **Create Axiom Form Data Source > Line Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight D3:J5) and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows:



Component properties

You can define the following properties for a Line Chart component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Selected Label Selected Series	The currently selected data point in the chart. This is identified by the corresponding label for the data point (the XValueName) and the Series that the data point belongs to. These settings are only used if the chart is configured to support interactivity. These settings serve two purposes: • They specify the initially selected data point of the chart, when the user first opens the form. You can leave the settings blank to have no initial selection, or you can enter an XValueName from the data source into the Selected Label field, and the corresponding Series name into the Selected Series field. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference these settings. • When a user views the form and selects a data point in the chart, the XValueName and Series name of the selected point will be submitted back to the source file and placed in these cells on the Form Control Sheet. Formulas can reference these cells in order to dynamically change the form based on the currently selected data point in the chart.
Auto Submit	Specifies whether the Axiom form is automatically refreshed when a user selects a data point in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will refresh unnecessarily if the user clicks on data points in the chart. If enabled, then the form automatically refreshes when the user selects a data point in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.

Item	Description
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Column Chart on the canvas, then the default is automatically set to Column. You can change the default chart type by changing this value.

Item	Description
Composition Kind	Specifies the composition of items in the chart when multiple series are present: • **Side by Side** (default for Bar, Column, and Line Chart components): Bars, columns, and lines for each series are displayed side-by-side. For area charts, the areas overlap the same space. **NOTE:** If you choose this option for an area chart, you may also want to set the **Area Series Opacity** to **Translucent**, so that you can see the detail for overlapping areas. • **Stacked** (default for Area Chart components): Series are stacked in a single bar or column. For area and line charts, each area or line is stacked on top of each other. The selected composition kind applies to all series in the chart.
Area Series Opacity	Specifies the opacity of area series within the chart: • **Opaque** (default): Area series are opaque. • **Translucent**: Area series are translucent. This is typically selected if the **Composition Kind** of the chart is set to **Side by Side**, so that you can see all areas in the chart. This setting is ignored for all other series kinds.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the XValueNames defined in the data source, and the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.

Item	Description
Name Rotation	The degree of rotation for the chart names (the XValueNames from the data source). By default this is blank, which means that the names are not rotated. To rotate the names, enter a value from -360 to 360. The purpose of this setting is to allow displaying longer names as vertical or slanted. For example, a value of -45 displays the name as slanted upward, whereas a value of 45 displays the name as slanted downward.  *-45 degree name rotation*  *45 degree name rotation* **NOTE:** For bar charts, the names run down the Y-axis instead of along the X-axis as shown here.
Primary Y-Axis Label	Optional. The label for the primary Y-axis. This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions". **NOTE:** If the chart is a bar chart, then the axes are flipped. XValueNames from the data source are displayed along the traditional Y-axis (down the side of the chart), whereas Y-axis labels are displayed along the traditional X-axis (across the width of the chart).
Primary Y-Axis Format	Optional. Specifies the format for the primary Y-axis values: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Decimals	Optional. Specifies how many decimal places to show on the primary Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Primary Y-Axis Min Primary Y-Axis Max	Optional. Specifies the maximum value and the minimum value for the primary Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Primary Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
Use Secondary Y-Axis	Select this option if you want to create a chart with two different Y-axis scales. If this check box is selected, then another series of Y-Axis settings will display for the Secondary Y-Axis. These settings work the same way as the settings for the Primary Y-Axis. Typically, multiple Y-axis scales are only used with combination charts, meaning charts with two types of series. For more information, see Creating combination charts.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the XYChart family. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Line Chart component can be set up to allow the user to select a data point on a line. The selected item is submitted back to the source file, and written to the **Selected Label** and **Selected Series** settings on the Form Control Sheet, using the XValueName and the corresponding Series name.

If you want the Axiom form to respond to the currently selected data point, then you must set up the file so that another component references the selected label and/or series and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

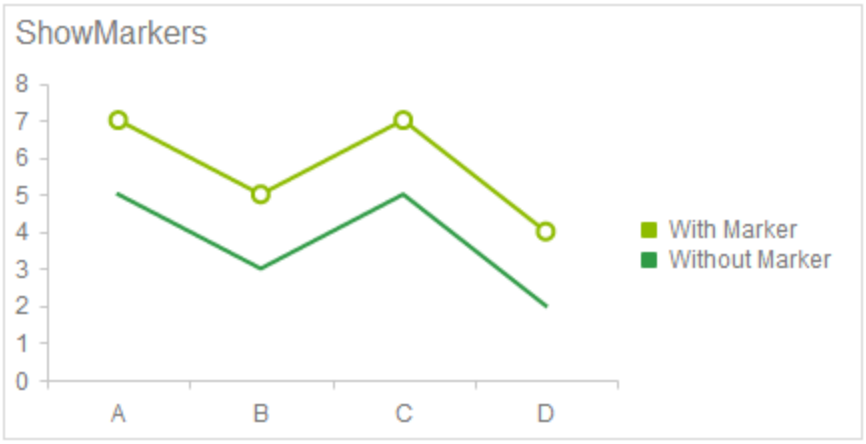
Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular line in the chart.

Example

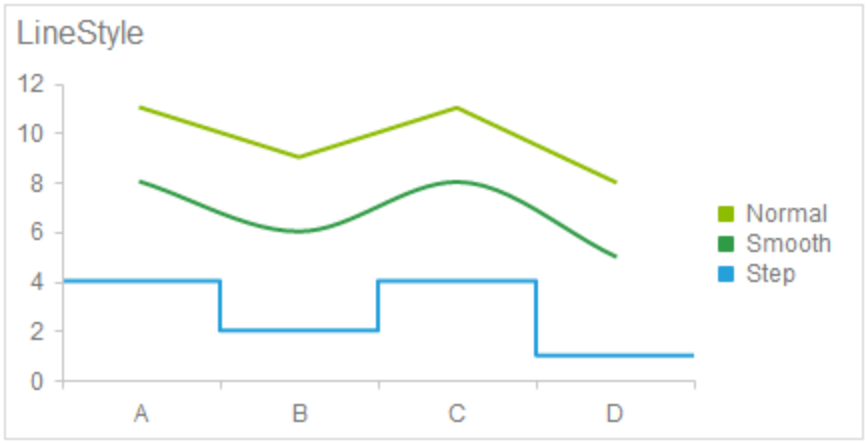
The Axiom form could contain a line chart that shows budget and actuals data by month. If you want users to be able to see the details about the data in any particular month, you could set up a second chart that references the selected label and series of the first chart. For example, if the user selects the Budget line for February in the first chart, the second chart will be updated to show detailed budget data for February. The second chart could support additional interactivity so that the Axiom form user can decide how they want to view this detailed budget data (for example, broken out by account category or by department regions).

Using optional line settings

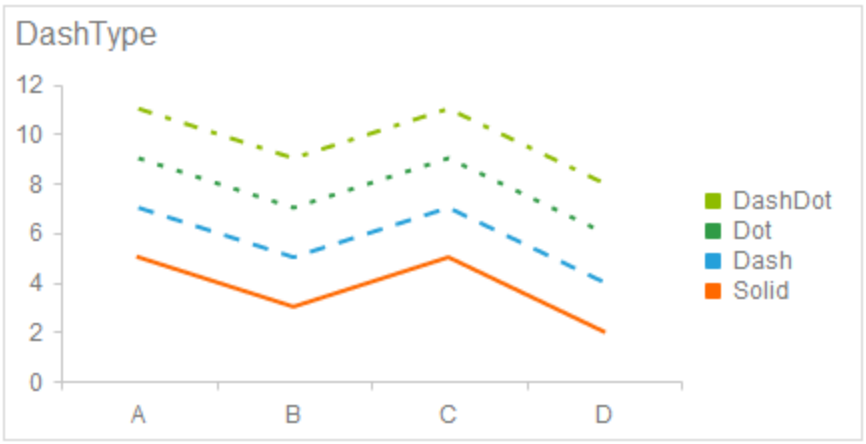
The XYChart data source supports several optional settings for line series: `[ShowMarkers]`, `[LineStyle]`, and `[DashType]`. The following screenshots show examples of these various options.



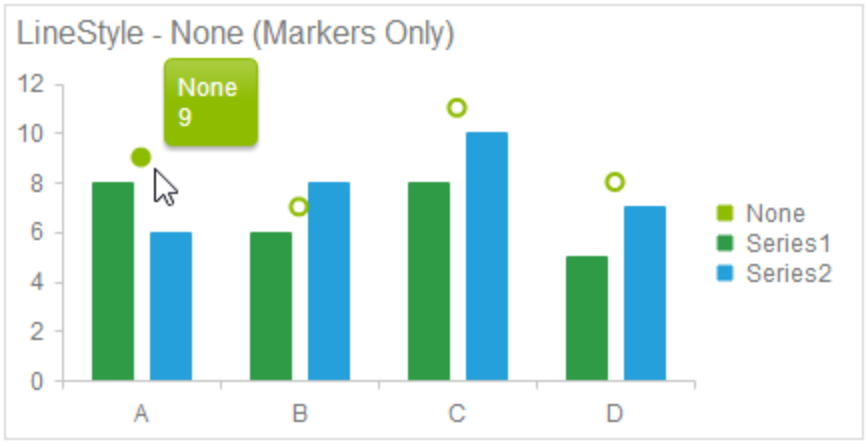
Example with markers enabled and disabled



Example line styles (None is not shown)



Example dash types



Example of None line style showing markers only, combined with bar series

Linear Gauge component

The Linear Gauge component for Axiom forms displays a value along a defined standard of measurement. Gauges can have up to three defined ranges of values. Typically, gauge ranges are used to differentiate "good" values versus values that are less desirable or that may indicate trouble.

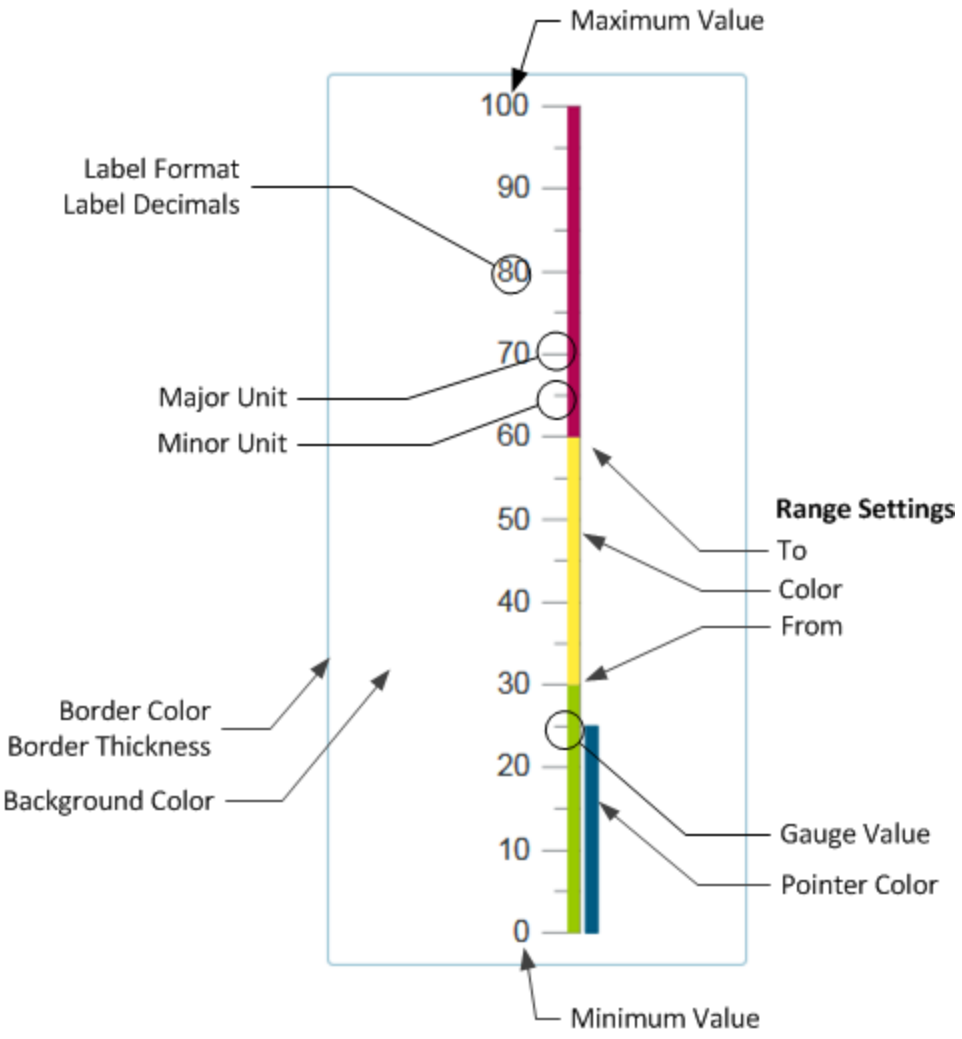
Bullet charts and gauges are visually similar, but are typically used for different purposes. Although both components display a value along a defined measurement scale, the bullet chart adds the concept of a target value and therefore explicitly communicates performance against a defined goal. The overall appearance of bullet charts is also more streamlined than gauges, which are often styled to resemble real-life measurement tools such as thermometers or speedometers.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

There are two types of gauges available for Axiom forms: Radial and Linear. Both gauges use the same properties to define the gauge, but the display of these properties is different. A radial-style gauge has an appearance similar to a car speedometer, whereas the appearance of a linear style gauge is similar to a thermometer. For more information on radial-style gauges, see Radial Gauge component .

Component properties

You can define the following properties for a Linear Gauge component. The following screenshot shows an example gauge with the major properties that impact the display:


Maximum Value
100
90
80
70
60
50
40
30
20
10
0
Minimum Value
Label Format
Label Decimals
Major Unit
Minor Unit
Range Settings
To
Color
From
Border Color
Border Thickness
Background Color
Gauge Value
Pointer Color

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Gauge Kind	Specifies the kind of gauge: • **Radial**: The measurement ranges are displayed in a circular pattern. The measured value is indicated by a pointer that originates from the center of the circle with the end resting on the measured value. • **Linear**: The measurement ranges are displayed in a straight line. The measured value is indicated by a second bar that starts at the lowest value of the gauge and continues until the reaching the measured value. The gauge kind is selected by default based on what type of gauge you placed on the canvas. You can switch the gauge type using this option.
Gauge Value	The measured value for the gauge. This is the value that will be indicated by the gauge pointer. If you do not specify a value, then the gauge pointer will be at the minimum value for the gauge. **NOTE:** If the gauge value is outside of the defined scale of measurement for the gauge, the pointer will be set to the minimum or maximum value of the gauge as appropriate (depending on whether the value exceeds the maximum value or is lower than the minimum value).
Minimum Value	The minimum value for the gauge scale of measurement. By default this is 0.
Maximum Value	The maximum value for the gauge scale of measurement. By default this is 100.
Minor Unit	Interval at which tick marks should display on the gauge to indicate values along the measurement scale. By default this is 5.
Major Unit	Interval at which number labels should display on the gauge to indicate values along the measurement scale. By default this is 20. The minimum value and the maximum value are always labeled on the gauge.
Label Format	Specifies the format for the major unit labels: **Number** (default), **Currency**, or **Percent**.
Label Decimals	Specifies the number of decimals to display on the major unit labels. By default, no decimal places are shown (0).

Item	Description
Labels Are Outside Gauge	This option does not apply to linear gauges. If enabled for a linear gauge, it will be ignored. By default, this option only displays on the Form Control Sheet as part of the overall set of gauge properties. It is hidden from the Form Designer and the Form Assistant if you start with a Linear Gauge component. However, if you start with a Radial Gauge and then switch the Gauge Kind to Linear, the option will continue to display but will be ignored. Labels for linear gauges always display to the left of the gauge if the orientation is vertical, and at the bottom of the gauge if the orientation is horizontal.
Orientation	Specifies the orientation of the gauge: **Vertical** (default) or **Horizontal**.
Pointer Color	The color of the pointer. If left blank, the pointer color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

Range settings

You can define up to three ranges for the gauge. Ranges are defined by a starting and ending value, and a color to shade that range. If you do not want to use a particular range, leave the settings for that range blank.

If you want the ranges to be continuous, then the **To** value of one range and the **From** value of the next range should be the same number. For example, if range one is from 0 to 20, then the from value for range two should be 20.

Range colors can be inherited from the style or skin (in that order), or colors can be manually specified. By default, all platform skins are set to use green, yellow, and red.

Item	Description
Range 1 From	The starting value of the range.
Range 1 To	The ending value of the range.
Range 1 Color	The color for the range.
Range 2 From	The starting value of the range.

Item	Description
Range 2 To	The ending value of the range.
Range 2 Color	The color for the range.
Range 3 From	The starting value of the range.
Range 3 To	The ending value of the range.
Range 3 Color	The color for the range.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Even though Pointer Color and the range colors can be affected by styles, these properties are exposed as component behavior properties because they are unique to the gauge component type. Also, the Axiom Software platform does not currently provide any styles specifically designed for gauge components.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Map View component

The Map View component displays geospatial data on a map. The component can be used to plot comparative data on a map (for example, using bigger circles to indicate greater revenue at a location), or the component can serve as an geographical selection tool to interactively display additional data based on the user's selected point on the map.

Defining a map view is a multiple-step process that requires the following:

- Creation of a MapView data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Map View component on the Axiom form canvas.
- Optional. Import of a GEO Feature file into the Reports Library to provide mapping shapes and feature data. For more information, see Using GEO Feature Files and Map Tiles with Map View components.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

A Map View component must have a defined data source within the report to indicate the data for the map. Each row in the data source is used for one of the following purposes:

- To define a specific location on the map, indicated by the latitude and longitude of the location. This location can be marked on the map using a pin or a circle. Circles can display in relative size based on some aspect of the location.
- To define properties for a feature (shape) displayed on the map, as defined in the GEO Feature File used by the Map View component. For example, if the GEO Feature File defines U.S. states, then each state can optionally be defined in the data source to set certain properties about the state on the map.

Depending on the purpose of your map view (and whether or not you are using a GEO Feature File), your data source may have a mixture of both types of rows, or just "location" rows, or just "feature" rows. For example, you might display various locations on a map view using either pins or relative circles and not use a GEO Feature File at all. Or at the other extreme you might only display "features" on a map so that users can select the features to view associated data (like state data), and not plot any specific locations at all. For more information, see Using GEO Feature Files and Map Tiles with Map View components.

The tags for the data source are as follows:

Primary tag

[MapView;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a Map View component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a row of data to display in the chart.

Column tags

[MarkerType]

Determines how the row of data will display in the map, as `Pin`, `Circle`, or `Feature`. The Pin simply indicates the location on the map using a pin-style graphic. The Circle both indicates the location on the map and can convey comparative data by varying the circle size or color based on some aspect of the location. Both Pin and Circle rows require a specific latitude and longitude in order to plot the location of the pin or circle on the map.

The Feature relates to a "feature" (shape) defined in the GEO Feature File as specified in the Map View component settings. By including the feature in the data source, you can define various display properties for that feature. Latitude and longitude are not required for Feature rows; the feature's shape and location is determined by the GEO Feature File.

NOTE: GEO Feature Files can also contain "point" features that define a single geographical location, which will be represented as pins on the map. However, when adding these items to the data source, the marker type must still be Feature.

[RowID]

An ID that uniquely identifies each row in the data source. For Pin or Circle rows, the ID can be any value, such as numbers or names. For Feature rows, the ID must correspond to the property specified in the Map View component settings as the **Feature Property ID**.

When a user selects a location or feature in the map, the value in the `[RowID]` column is written back to the **Selected Value** field for the Map View component. Formulas can reference this value to change or display something in the form based on the selected value—for example to change the `[FillColor]` for the selected item, or to filter the data shown in another chart based on the selected item.

[Lat]

The latitude of the location to display in the map, in decimal degrees. For example, 45.523452 is the latitude in decimal degrees for Portland, Oregon.

For Pin or Circle rows, this value is required to define the location of the marker on the map. For Feature rows, this value is optional. If defined for a Feature row, it is used to define the center point of the feature for purposes of placing the label text.

[Lon]

The longitude of the location to display in the map, in decimal degrees. For example, -122.676207 is the longitude in decimal degrees for Portland, Oregon.

For Pin or Circle rows, this value is required to define the location of the marker on the map. For Feature rows, this value is optional. If defined for a Feature row, it is used to define the center point of the feature for purposes of placing the label text.

[MarkerSize]

For Circle rows, this value defines the size of the circle in pixels. In most cases, the size of the circle should be based on some aspect of the location in order to display relative data—such as sales per location, with a higher number of sales displayed using a larger circle. However, you can also display all circles with the same size, as an alternative to using a pin to simply mark the location on the map.

For Pin or Feature rows, this value can be used as an alternate way of setting the `[LabelBoxSize]` (see optional column tags). It does not affect the size of the pin or the feature.

[FillColor]

The color of the circle or the feature. You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua).

If omitted for a Circle row, then the circle will use the **Text Color** specified in the Map View component settings. If omitted for a Feature row, then the feature will use the **Feature Fill Color** specified in the component settings. This setting does not apply to Pin rows (or to point-style features); pins always display in blue.

You might use the `[FillColor]` column to specify different colors for each circle or feature (or for certain groups of circles or features), and/or you might use a formula to change the color for the currently selected item (by using a formula that references the **Selected Value**).

[LabelText]

The label text to display for the pin or circle location, or on the feature shape. This can be omitted if it is not necessary for your map. You can specify additional properties for the label such as color, position, and font size using the optional column tags.

Column tags (optional)

The following optional tags are not added by the data source wizard; you must manually add them to the data source as needed.

[PopupText]

Text to display in a popup text box when a user clicks on the pin, circle, or feature. For example, instead of using `[LabelText]`, you may want to display the label as a popup.

> **IMPORTANT:** If popup text is defined, then auto-submit is *disabled* for the item and no refresh will occur when the item is selected. You will not be able to reference the selected item to trigger changes in the form. The only thing users can do with the item is click on it to read the popup text.

[Href]

A URL to open in a new window when the pin, circle, or feature is selected in the map. You can use functions such as GetFormDocumentURL or GetDocumentHyperlink to generate hyperlinks to open Axiom forms or other Axiom files.

[SeriesName]

The name of the series. This can be used to group multiple rows in the data source by assigning them the same series name. Currently the Map View component does not do anything with this information, but you can use it in the data source to set other properties to the same value for all rows belonging to the group—for example, to set the `[FillColor]` the same for all series that use the same series name.

[LineColor]

For Feature rows, this value defines the color of the line used to draw the feature shape. For Circle rows, this value defines the border color for the circle. You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua).

If omitted, the **Text Color** specified in the component settings is used. This value does not apply to Pin rows.

[LineSize]

For Feature rows, this value defines thickness of the line (in pixels) used to draw the feature shape. If omitted, the **Feature Line Width** specified in the Map Component settings is used.

For Circle rows, this value defines the thickness of the border (in pixels) for the circle. If omitted, the circle does not have a border.

This value does not apply to Pin rows.

[DashSize]

Determines whether a dashed line should be used for Feature rows and Circle rows. To display dashed lines, specify the gap between dashes in pixels. If omitted, lines are solid. This value does not apply to Pin rows.

[LabelColor]

The color of the label text. You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua). If omitted, the color will be determined by the **Text Color** in the Map View component settings.

[LabelPosition]

The position of the label text. This is based on a 9-point grid similar to a touch-phone keypad, with 1 as the top left position, 5 as the middle position, and 9 as the bottom right position. If omitted, the default is 5 (center).

For Pin and Circle rows, the position is relative to the marker (treating the marker as the middle of the 9-point grid). For Feature rows, the feature shape is treated as if the 9-point grid is overlaid on the shape. If a `[Lat]` and `[Lon]` have been defined for a Feature row, then that location will be used as the center point of the grid. Some trial and error with `[LabelPosition]` and `[LabelBoxSize]` may be necessary to find a good position within certain feature shapes (or specify a `[Lat]` and `[Lon]` to place the label at a specific location within the feature shape).

[LabelBoxSize]

Defines an alternate size for the label position points, in pixels. Larger box sizes mean that the position points will be farther away from each other and from the relative center.

[FontStyle]

The style of the font used for the label text, either `Default` (the style determined by the form-level skin), `Normal`, or `Italic`. If omitted the default style is used.

[FontSize]

The size of the font used for the label text, in pixels. If omitted the size is determined by the form-level skin.

[FontFamily]

The font family used for the label text. If omitted, the system default for sans serif font will be used. If you specify a font family, it is strongly recommended to use a very common, basic font such as Arial, which all client machines and devices are likely to support.

[FontWeight]

The weight of the font used for the label text, either `Default` (the weight for the form-level skin), `Normal`, or `Bold`. If omitted the default weight is used.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows simple mapping data flagged in a report. In this example the data source contains both pin locations and features from the GEO Feature File.

	A	B	C	D	E	F	G	H	I	J
2										
3		[MapView;Countries]		[RowID]	[Lat]	[Lon]	[MarkerType]	[MarkerSize]	[FillColor]	[LabelText]
4		[Series]		United Kingdom			Feature			
5		[Series]		France			Feature			
6		[Series]		Spain			Feature			
7		[Series]		Belgium			Feature			
8		[Series]		Ireland			Feature			
9		[Series]		Sweden			Feature			
10		[Series]		London	51.50722	0.1275	Pin			
11		[Series]		Manchester	53.48095	-2.23743	Pin			
12		[Series]		Edinburgh	55.95206	-3.19648	Pin			
13		[Series]		Paris	48.85341	2.3488	Pin			

To use the Data Source Wizard to add the tags, right-click a cell and select **Create Axiom Form Data Source > Map View**. You can right-click a single empty cell to place the initial tags and then fill out the data, or you can have the data already in the spreadsheet and highlight the applicable data to add the tags. The cells in the row above the data and the column to the left of the data must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows. This example assumes use of a Map Tile Provider in addition to the GEO Feature File that defines the country shapes (drawn with a dark gray outline here).

Component properties

You can define the following properties for a Map View component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the component. You must have defined the data source within the report using the appropriate tags in order to select it for the component. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.

Item	Description
Selected Value	The currently selected item in the map (a pin, circle, or feature). This setting serves two purposes: • It specifies the initially selected item in the chart, when the user first opens the form. You can leave the setting blank to have no initial selection, or you can enter an ID as defined in the `[RowID]` column of the data source (for feature items, this is the value for the property specified in the **Feature ID Property**). • When a user views the form and selects an item in the map, the ID of the selected item will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected item in the map. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected value to be read from and written to the specified cell reference instead of directly within the Selected Value cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected value is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).

Item	Description
Lat, Lon, Zoom	The starting latitude, longitude, and zoom level for the map. The specified location will be centered in the middle of the component area, displayed at the specified zoom level. Each value must be entered in the stated order, separated by commas. The latitude and longitude must be expressed in decimal degrees. The zoom level is an integer from 1-18, where 1 is fully zoomed in and 18 is fully zoomed out. By default, the latitude and longitude are set to the middle of the United States. One good approach to finding the best starting value for your map is to enable **Show Map Params** and **Allow Map Navigation** and then preview your form. Center and zoom the map as desired, and then highlight and copy the values displayed in the bottom left corner of the component. You can then copy these values into the **Lat,Lon,Zoom** field. These values are only applied when the form is first opened. When the form is refreshed using a Button component or auto-submit behavior on an interactive component, the map does *not* return to these starting coordinates; instead the user's current location is retained.
Show Map Params	Specifies whether the current map parameters—latitude, longitude, and zoom level—will display in the bottom left corner of the form. By default this is disabled, which means the parameters will not display.
Allow Map Navigation	Specifies whether users can move and zoom the map. By default this is enabled, which means: • Zoom in and out icons will display on the map so that users can change the zoom level. • Users can click and drag to move the current map location within the component space. If this option is disabled, then the map display is fixed at its starting **Lat,Lon,Zoom** setting.
Map Tiles Provider	The provider of map tiles to display in the component. By default this is set to **None**, which means no map tiles will display. You must specify either a Map Tiles Provider or a GEO Feature File URI (or both) to define the map graphic for the Map View component. For more information, see Using GEO Feature Files and Map Tiles with Map View components.

Item	Description
GEO Feature File URI	The file that contains the set of GeoJSON "features" (geographical shapes) to display in the component. The file contents must use the GeoJSON specification and the file type must be JSON. Click the **[...]** button to select the file. The file must be located in the Reports Library. If the file is not already saved in the Reports Library, you can right-click a folder and select **Import** to import the file (if you have the appropriate rights to do so). You must specify either a Map Tiles Provider or a GEO Feature File URI (or both) to define the map graphic for the Map View component. For more information, see Using GEO Feature Files and Map Tiles with Map View components. If you specify a GEO Feature File, then several optional feature properties become available, as detailed in the following table.
Title Text	The title text for the component. This text displays in the title bar of the component panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text does *not* display. In the latter case you can alternatively use a separate Label component to create a chart title. **NOTE:** The font type / size / weight / style of the title text are dependent on the form-level skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.

Feature properties

The following optional feature properties are only available if a GEO Feature File has been specified.

Item	Description
Feature Line Width	The width of the feature lines defined by the specified GEO Feature File, in pixels. This setting is only used if no size is defined for a feature in the `[LineSize]` column in the data source. A line width of 0 means no feature lines will display on the form. If left blank, and if `[LineSize]` is not being used, then the line width is determined by the style or skin (in that order).
Feature Fill Color	The fill color of the features defined by the specified GEO Feature File. This setting is only used if no fill color is defined for a feature in the `[FillColor]` column of the data source. If left blank, and if `[FillColor]` is not being used, then the fill color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.
Feature ID Property	The property in the specified GEO Feature File that identifies each individual feature. Enter the name of a property defined in the file. Only applies if a **GEO Feature File URI** is specified. **NOTE:** This setting is case-sensitive and must match the property in the GEO Feature File exactly. For example, if the property is defined as "code" in the file then you must enter "code" into the Feature ID Property; "Code" will not be recognized. This property is used to enable interactivity for the map features. When a feature is selected in the map, the property value for the selected feature will be written to the **Selected Value** field for the Map View component. The form can then be configured to change in some way based on the selected feature. The property can also be used in the `[RowID]` for the data source, so that you can define formatting for each feature. If this setting is left blank, then the features in the form will not be selectable, and you will not be able to define formatting for the features in the data source.

Item	Description
Feature Description Property	The property in the specified GEO Feature File that contains descriptive text that you want to display as popup text when the user clicks on the feature. Enter the name of a property defined in the file. This setting only applies if a **GEO Feature File URI** is specified, and the description is only displayed for features that are *not* defined in the data source. Additionally, **Filter Unmatched Features** must be disabled. **NOTE:** This setting is case-sensitive and must match the property in the GEO Feature File exactly. For example, if the property is defined as "name" in the file then you must enter "name" into the Feature Description Property; "node" will not be recognized. **IMPORTANT:** If a property name is specified here, then auto-submit is *disabled* for affected items and no refresh will occur when these features are selected. You will not be able to reference these items to trigger changes in the form. The only thing users can do with affected items is click on them to read the popup text.
Filter Unmatched Features	Specifies whether only matched features from the GEO Feature File will be shown in the map view. By default this is not selected, which means that all features listed in the GEO Feature File will display in the map view. If selected, then the map view will only display features that are also listed in the MapView data source, based on matching the `[RowID]` in the data source to the value in the file (for the specified **Feature ID Property**).

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Even though Feature Line Width and Feature Line Color can be affected by styles, this property is exposed as a component behavior property because it is unique to this component type. Also, Axiom Software does not currently provide any styles specifically for maps.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For

more information on these properties, see Position and size properties.

Interactive behavior

The Map View component can be set up to allow the user to select an item in the map, such as a pin, circle, or feature. The selected item is submitted back to the source file, and written to the **Selected Value** setting on the Form Control Sheet, using the value defined in the `[RowID]` column of the data source.

NOTES:

- Feature items are not required to be set up in the data source in order to enable interactivity. The ID for all feature items is determined by the **Feature ID Property**. This setting identifies a property in the GEO Feature File that defines the ID for the feature items.
- Map View components always auto-submit in response to changing selections in the map; it is not possible to disable the auto-submit behavior.

If you want the Axiom form to respond to the currently selected item, then you must do either or both of the following:

- Set up the file so that something else in the file references the selected value and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.
- Define a URL in the `[Href]` column of the data source. If a URL is defined here, it will be opened in a new window when the corresponding item is selected in the map.

Example

The Axiom form could contain a map with pins for store locations. If you want users to be able to see more details about a particular store, the form could also contain a formatted grid that displays data about the currently selected store. A filter can reference the Selected Value field, so that the Axiom query or GetData functions shown in the formatted grid only show the appropriate data.

Using GEO Feature Files and Map Tiles with Map View components

When using a Map View component, the actual map graphic displayed on the Axiom form is defined by one or both of the following items:

- A **GEO Feature File** can be used to define "features" (shapes) to display on the map view. For example, the GEO Feature File can define the shapes of countries, states, or counties, or any geospatial construct such as the sales territories used by your organization or a representation of the London tube lines. These features can display on the map "as is," or you can add feature rows to your MapView data source to define display properties for the features.

- A **Map Tiles Provider** can be used to display a background map tile on the map view. The level of detail on the map tile depends on the selected provider. The map tile is always displayed "as is" and cannot be formatted by the data source or selected for interactivity.

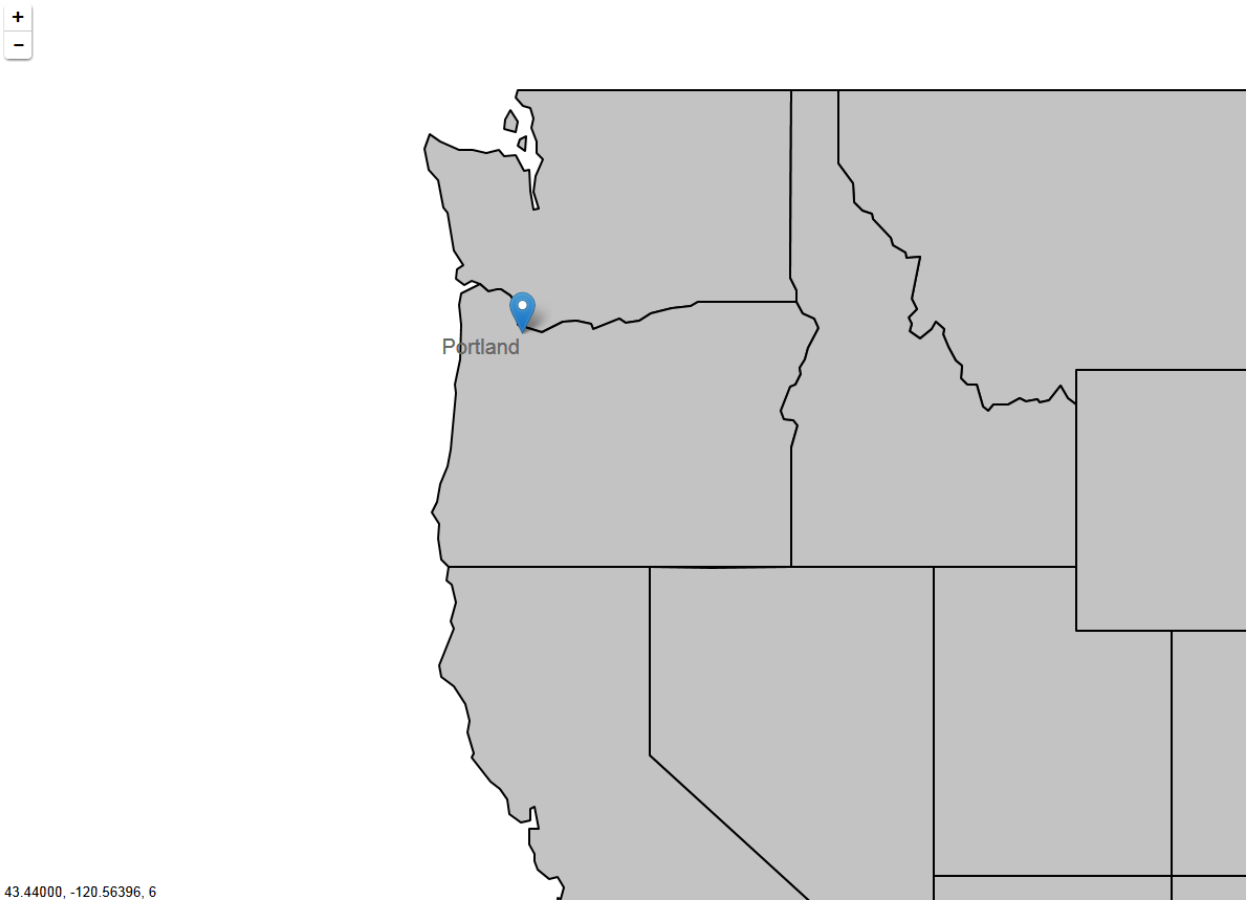
Deciding how to define your map graphic

When deciding how to define the map graphic for a Map View component, you should consider factors such as the following:

- **How much detail do you need to display on the map?** For example, if you want users to be focused solely on the map markers and features, then you may want to use only a GEO Feature File so that users aren't distracted by extraneous details on an all-purpose map tile such as cities, roads, and rivers.
- **Do users need to select feature areas on the map, or just plotted locations?** If users only need to be able to select specific plotted locations on the map, then you can choose to use only a map tile as the map graphic. However, if users need to be able to select specific feature areas such as a state or a country, then you must use a GEO Feature File to define these features and make them selectable on the map.
- **Are the features you need to display available on an all-purpose map?** Map tiles generally display standard geographical features and governmental entities like rivers, lakes, states, and cities. If the features you need to display on the map are more specialized, such as your organization's specific sales territories, then you must use a GEO Feature File to define the parameters of these features.
- **How important is it that the map be cosmetically appealing?** Map tiles give the map graphic a lot of color, depth, and other visual interest. If it is very important that the map graphic "looks attractive" versus just being useable, you may want to use a Map Tile either alone or in conjunction with a GEO Feature File.

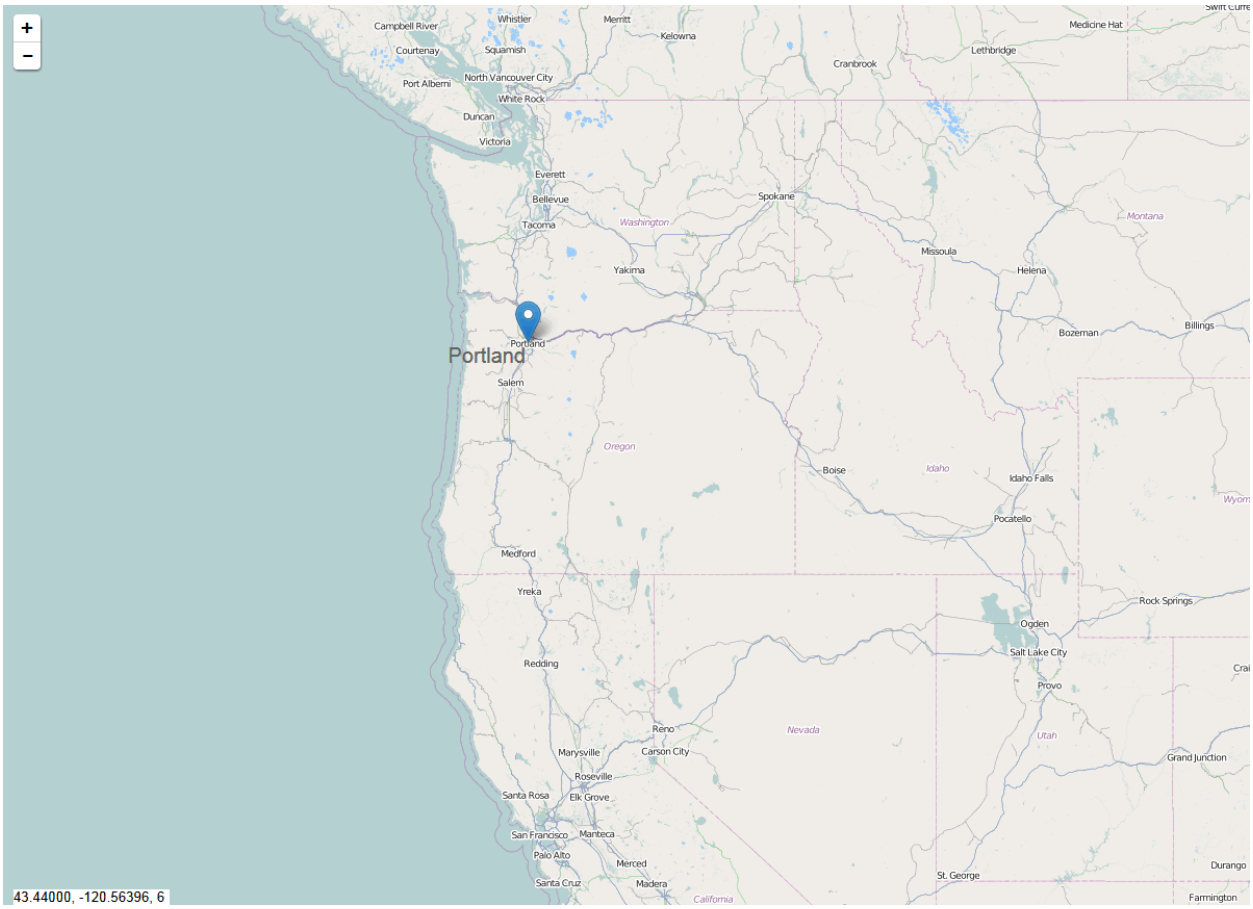
The ability to zoom in and out and move the map graphic is available regardless of which approach is chosen.

The following example screenshots illustrate how Map View components can look and behave differently based on whether the component only uses a GEO Feature File, or only a Map Tiles Provider, or both. In these screenshots, one example location (a Pin row) is defined in the data source.



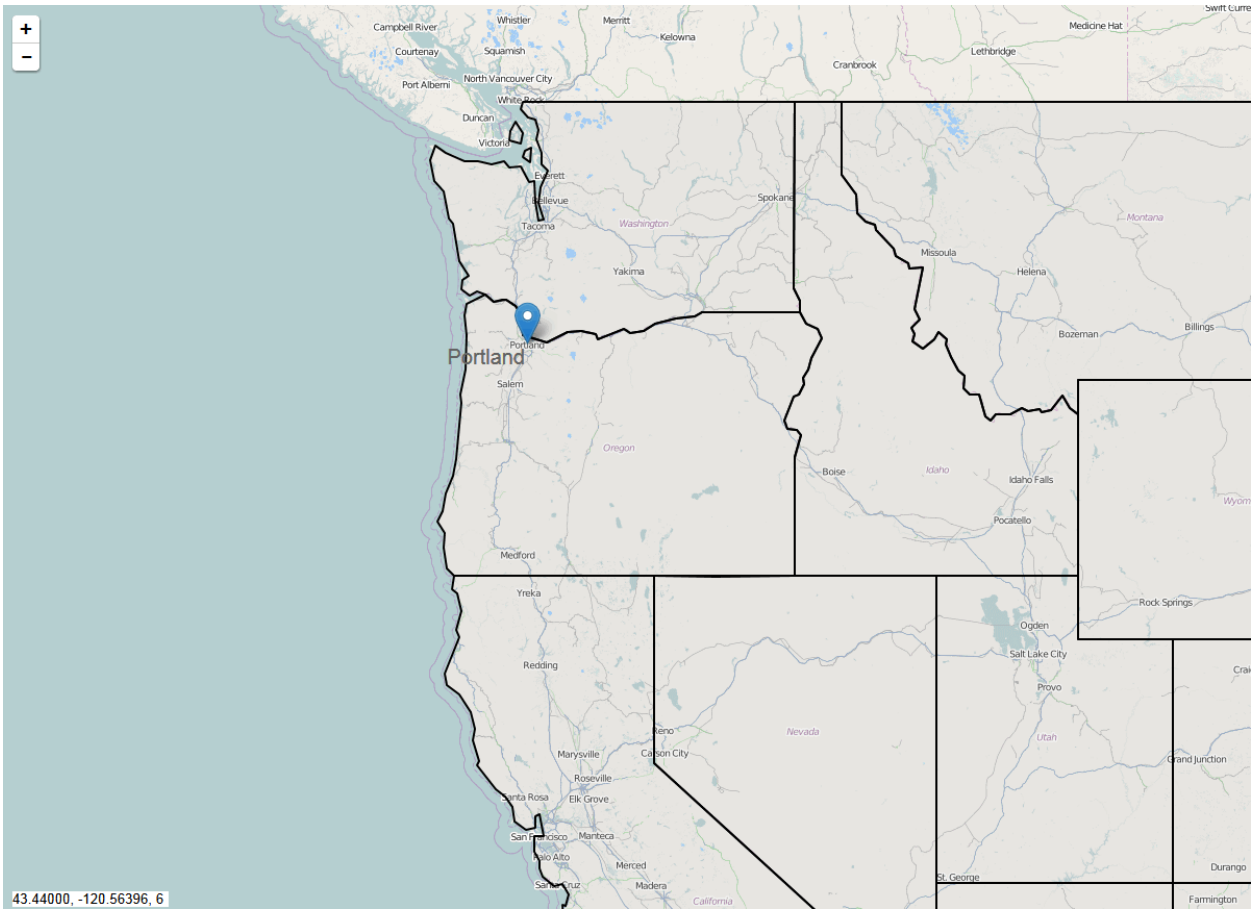
GEO Feature File only

In this example, a GEO Feature File is used to define the state "features" (shapes). No Map Tile provider is specified. When using this approach, the map can be interactive based on selecting plotted location points from the data source (as shown using a pin in this example) or by selecting individual state shapes. You can also define display properties for the state features such as fill color, border color and thickness, label text, etc.



Map Tiles Provider only

In this example, a Map Tile provider is specified to display the map graphic. No GEO Feature File is specified. When using this approach the map can be interactive based on selecting plotted location points from the data source only (as shown using a pin in this example). The background Map Tile itself cannot be selected and its formatting cannot be modified. However, the map details are provided automatically and display the plotted location within an easily recognizable context.



GEO Feature File and Map Tiles Provider

In this example, both a GEO Feature File and a Map Tiles Provider have been specified. The features in the GEO Feature File overlay the map tile and can still be used as interactive elements in the form.

Using a GEO Feature File

The GEO Feature File for Map View components uses the GeoJSON format to define the "features" (shapes) to display on the map. GeoJSON is an open standard format for encoding sets of geographic data structures. For more information on GeoJSON, visit http://geojson.org/.

You can obtain a GeoJSON file as follows:

- Axiom Software provides a set of standard files that you can use as is or customize to your needs. These standard files are located at `\Axiom\Axiom System\GEO Feature Files`. These files can be used to provide the following shapes:
  - U.S. States
  - World Countries

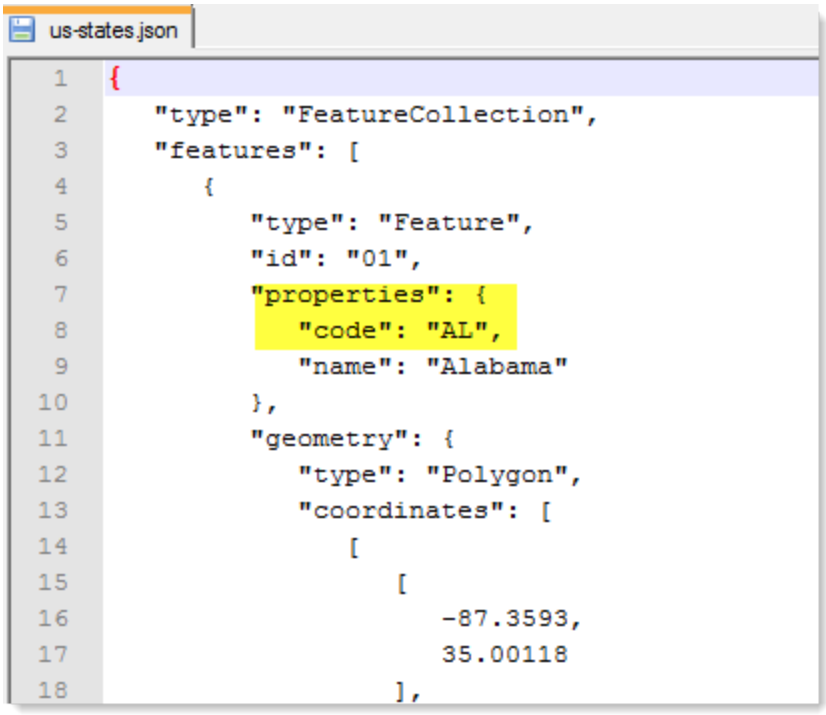
- Various organizations and web sites may make GeoJSON files available for public use.
- Your organization can create your own GeoJSON file using the specifications detailed on the official GeoJSON website linked above.

Once you have obtained the file, you must import it into the Reports Library, and then use the **GEO Feature File URI** setting in the Map View components to point to the file.

Enabling interactivity based on features in the GEO Feature File

If you want the features in the file to be interactive (selectable on the map), then you must complete the **Feature ID Property** in the Map View component settings. This setting tells Axiom Software which property in the GEO Format File should be used to identify the currently selected feature. When a user selects a feature in the map, the value for that property will be written to the **Selected Value** field for the Map View component.

For example, the standard `us-states.json` file has a property named code that uses the two-letter code for each state.



To use the code property to identify the state features in this file, you would indicate `code` as the **Feature ID Property**. When a user selects the Alabama feature in the map, the code `AL` will be written to the **Selected Value** field. The form can be configured to respond to the current selection—for example, to open another form with information relating to this state, or to filter the data displayed in another component such as a formatted grid or a column chart.

Although it is not required to define the feature items in the MapView data source in order to enable this interactivity, in many cases you will want to do this anyway in order to define display properties for each

feature. When features are defined in the data source, the `[RowID]` for those features must use the property specified as the Feature ID Property. See the following section for more information.

Formatting features from the GEO Feature File in the data source

If desired, you can add the features from the GEO Feature File to the MapView data source, so that you can define display properties for the feature. You can define properties such as the fill color, line thickness and color, label text and position, and font properties. You can also specify a hyperlink to launch when the feature is selected, such as to open another form with information about the selected feature.

To do this, add a row for each feature that you want to style. The `[RowID]` for the feature must use the same property specified as the **Feature ID Property** in the Map View component settings. For example, if the property used is `code` for two-letter state codes, then enter the two-letter state codes as the RowID.

For more information on the available properties that can be set for features in the MapView data source, see Map View component .

In most cases, GEO Feature Files are used to define shape features for the map. However, the GeoJSON specification also supports "point" features that relate to a single specific geographical location. If the GeoJSON file contains point features, these will be displayed using pins on the map. In the data source, point features must still be defined as Feature rows, not Pin rows—however they are subject to the same formatting limitations as Pin rows, such as not supporting a fill color.

Using a Map Tiles Provider

To use a Map Tiles Provider, select the desired provider from the **Map Tiles Provider** setting in the Map View component properties. Currently, the only available provider is Open Street Map.

When using a Map Tiles Provider, you do not have to specify what area of the map to display. This will be determined based on the starting **Lat, Lon, Zoom** specified for the Map View component (latitude, longitude, and zoom level), and the size of the Map View component on the canvas (a larger component means that more area of the map will display).

If **Allow Map Navigation** is enabled for the component, then users will be able to move the map around within the component space to view different areas of the map. This means that users can move to any area on the map, including areas where no locations are plotted and no features are displayed (if using a GEO Feature File as well).

Pie Chart component

The Pie Chart component displays data in a segmented circle, where the size of each "slice" is proportional to the size of the data point in relation to the overall total.

Defining a pie chart is a two-part process that requires the following:

- Creation of a PieChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Pie Chart component on the Axiom form canvas.

Pie charts can also support form interactivity, to change the contents of the Axiom form based on the currently selected slice in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

A Pie Chart component must have a defined data source within the report to indicate the data for the slices. The tags for the data source are as follows:

Primary tag

`[PieChart;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a Pie Chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[PieItem]`

Each row flagged with this tag defines a row of data to display in the chart.

Column tags

`[Label]`

The display label for each value in the chart.

`[Value]`

The corresponding value for each label. The value determines the size of the slice in the pie chart, proportional to the overall pie value.

[Color]

Optional. This column indicates the color assignment for each slice in the pie chart. If omitted, then colors will be dynamically determined based on the style or skin (in that order). This tag is not added by the data source wizard; you must manually add it to the data source if you need it. See Specifying chart colors.

[Explode]

Optional. This column indicates which slices should be "exploded" when the chart is rendered (True/False). These exploded slices will be slightly separated from the rest of the chart with white space, and extended slightly outward. You might use this column to emphasize one or more items in the chart. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

This option is not intended to support user interactivity; if you want slices of the chart to "explode" when a user selects them, then you should use the **Explode Selected** property of the component instead. This column is ignored if **Explode Selected** is enabled.

NOTES:

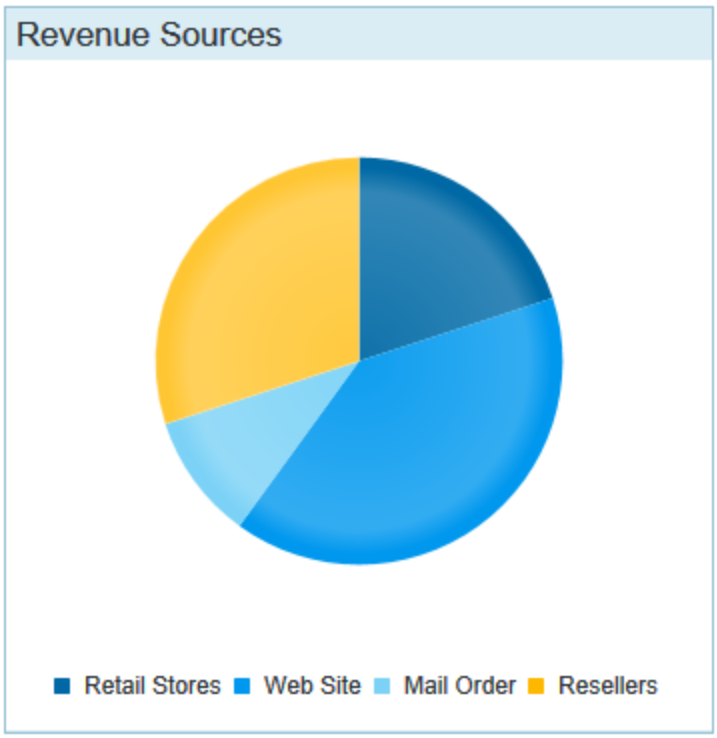
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows simple data flagged in a report. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D
1				
2		[PieChart;Revenue]	[Label]	[Value]
3		[PieItem]	Retail Stores	20%
4		[PieItem]	Web Site	40%
5		[PieItem]	Mail Order	10%
6		[PieItem]	Resellers	30%

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Pie Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight C3:D6) and then use the wizard. Axiom Software will add the tags as displayed in the example above. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows:



Component properties

You can define the following properties for a Pie Chart component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the report using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Pie Kind	The display type of the pie chart. You can specify **Pie** (default) or **Donut**.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If no legend is used, then the value for each slice will be noted on the pie chart, either using callouts or on the slices themselves (depending on whether **Place Labels on Slices** is enabled). If a legend is used, then labels are not placed on the chart and the user must mouse over each slice to see the value.

Item	Description
Selected Label	The currently selected pie slice in the chart. This setting serves two purposes: • It specifies the initially selected slice in the chart, when the user first opens the form. You can leave the setting blank to have no initial selection, or you can enter a Label name from the `[Label]` column of the data source. If **Explode Selected** is enabled, then the specified slice will be "exploded" by default in the form. • When a user views the form and selects a slice in the chart, the Label name of the selected slice will be submitted back to the source file and placed in this cell on the Form Control Sheet. Formulas can reference this cell in order to dynamically change the form based on the currently selected slice in the chart. **NOTES:** • This setting supports indirect cell references. You can enter a cell reference in brackets, such as `[Info!B3]`. This causes the selected label to be read from and written to the specified cell reference instead of directly within the Selected Label cell. • This setting supports use of the FormState tag and the SharedVariables tag, so that the selected order is stored in memory instead of written to the file, and therefore can be shared with other files. Form state can be used to share values between a form dialog and an active client spreadsheet, in the Desktop Client. Shared variables can be used to share values between multiple forms that are open in a shared form instance (composite forms).
Explode Selected	Specifies whether selected slices in the chart become "exploded" from the rest of the chart. These slices will be slightly separated from the rest of the chart with white space, and extended slightly outward. It is recommended to enable this item if the chart is set up to support interactivity, so that the user can clearly see which slice in the pie is selected. **NOTE:** If you want to explode certain slices of the chart for emphasis only, and have those items remain fixed (meaning, the user cannot change which slices are exploded by clicking on items), then you should not enable this option. Instead, you should use the `[Explode]` column in the data source. If this option is enabled, the `[Explode]` column is ignored.

Item	Description
Place Labels on Slices	Specifies where labels should display on the pie chart, if a legend is not being used. • If this option is selected, then labels display on the individual slices, within the pie chart. • Otherwise, labels display outside of the pie chart, using callout lines to link the labels to the appropriate slices. This setting only applies if **Legend** is set to **None**. If any particular slice is too small to legibly display the label, then you should disable this option. There is no way to configure a hybrid display (where labels display on large slices but callouts are used for small slices).
Auto Submit	Specifies whether the Axiom form automatically updates when a user selects a slice in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will update unnecessarily if the user clicks on items in the chart. If enabled, then the form automatically updates when the user selects a slice in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for pie charts. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Interactive behavior

The Pie Chart component can be set up to allow the user to select a slice in the chart. The selected slice is submitted back to the source file, and written to the **Selected Label** setting on the Form Control Sheet, using the name from the `[Label]` column of the data source.

If you want the Axiom form to respond to the currently selected item, then you must set up the file so that another component references the selected item and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

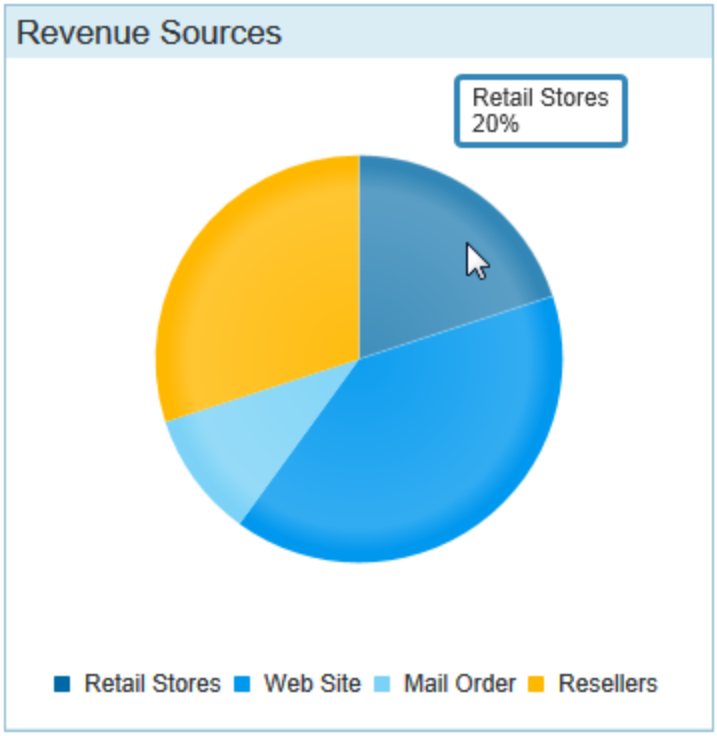
Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular slice in the pie chart.

Example

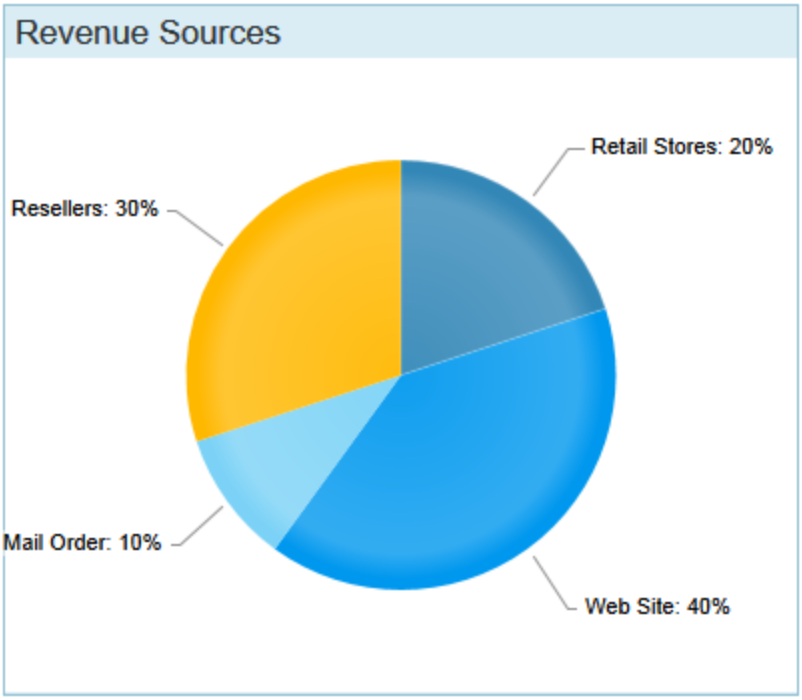
The Axiom form could contain a pie chart that shows a breakdown of revenue sources. If you want users to be able to see more details about the data in any particular slice, you could set up a second chart that references the selected label of the first chart. For example, if the user selects the Retail Stores slice in the first chart, the second chart will be updated to show revenue by individual store.

Pie Chart formatting notes

If a legend is enabled for the pie chart, then the values do not display directly on the chart. Users can hover their cursor over the pie slices to see the values, as shown in the following screenshot.



If no legend is present, then the pie labels and values are noted using callouts, as shown in the following screenshot. Alternatively, you can use the **Place Labels on Slices** option to display the labels and values on each individual slice instead of using callouts (however, this option is only recommended for pie charts with a small number of slices and short labels).



In some cases, if the callout text does not fit within the area allotted for the component, the callout may extend past the edge of the component and partially disappear. If this occurs, you can try the following options to correct the situation:

- Resize the component slightly larger or smaller. The size adjustment may cause the text to fit or allow the text to automatically wrap. (However, remember that if **Scale to Fit** is enabled for your form, then the issue may arise again as the form is dynamically sized larger or smaller.)
- If appropriate, you can manually place line breaks within the label text by adding the characters `\n` to the label text. For example, if the label text is defined as "Greater Metropolitan Area \n of Cleveland," then a line break will be placed after "Area." If line breaks are used, then automatic text wrapping is not applied to the label and only the marked breaks are honored.
- If the callouts still will not fit, then you may need to manually shorten the text by abbreviating the labels or changing the value format, or switch to using a chart legend instead.

In all cases, the format for the value is taken from the format used in the Values column of the data source, in the spreadsheet. For example, if the value is formatted as a percent in the spreadsheet, then it will display as a percent in the pie chart. If the value is formatted as a plain integer in the spreadsheet, or as a decimal, then it will display in that format in the pie chart.

Radial Gauge component

The Radial Gauge component for Axiom forms displays a value along a defined standard of measurement. Gauges can have up to three defined ranges of values. Typically, gauge ranges are used to differentiate "good" values versus values that are less desirable or that may indicate trouble.

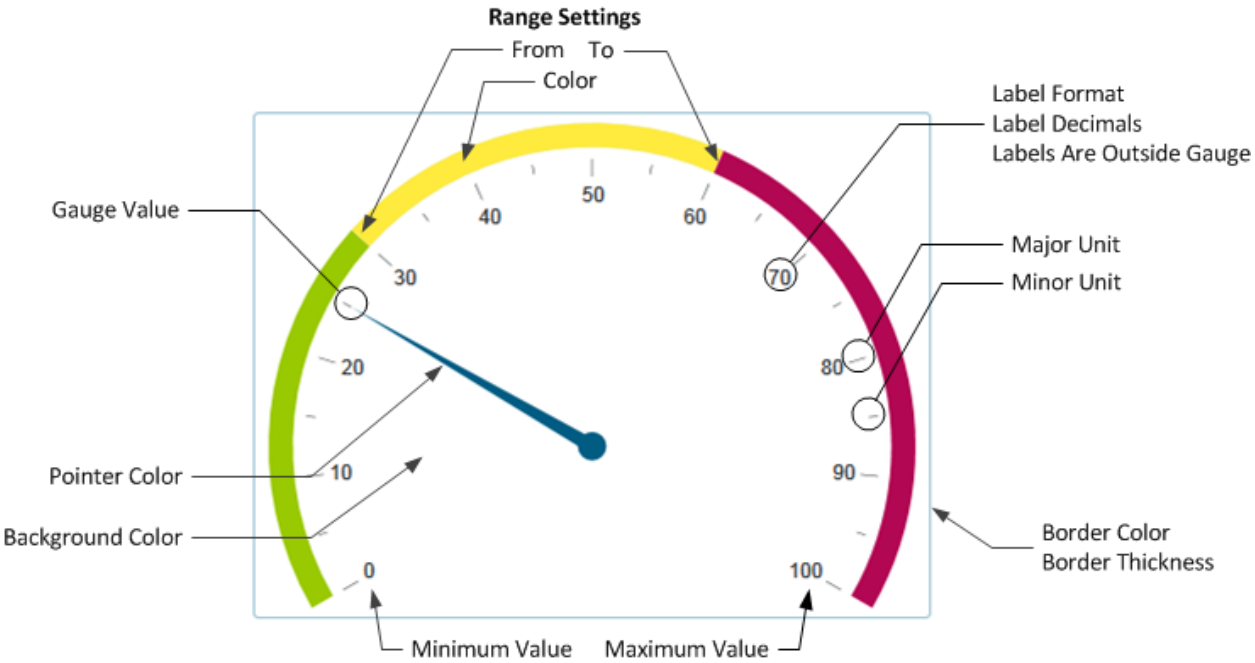
Bullet charts and gauges are visually similar, but are typically used for different purposes. Although both components display a value along a defined measurement scale, the bullet chart adds the concept of a target value and therefore explicitly communicates performance against a defined goal. The overall appearance of bullet charts is also more streamlined than gauges, which are often styled to resemble real-life measurement tools such as thermometers or speedometers.

There are two types of gauges available for Axiom forms: Radial and Linear. Both gauges use the same properties to define the gauge, but the display of these properties is different. A radial-style gauge has an appearance similar to a car speedometer, whereas the appearance of a linear style gauge is similar to a thermometer. For more information on linear-style gauges, see Linear Gauge component.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Component properties

You can define the following properties for a Radial Gauge component. The following screenshot shows an example gauge with the major properties that impact the display:



Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Gauge Kind	Specifies the kind of gauge: • **Radial**: The measurement ranges are displayed in a circular pattern. The measured value is indicated by a pointer that originates from the center of the circle with the end resting on the measured value. • **Linear**: The measurement ranges are displayed in a straight line. The measured value is indicated by a second bar that starts at the lowest value of the gauge and continues until the reaching the measured value. The gauge kind is selected by default based on what type of gauge you placed on the canvas. You can switch the gauge type using this option.
Gauge Value	The measured value for the gauge. This is the value that will be indicated by the gauge pointer. If you do not specify a value, then the gauge pointer will be at the minimum value for the gauge. **NOTE:** If the gauge value is outside of the defined scale of measurement for the gauge, the pointer will be set to the minimum or maximum value of the gauge as appropriate (depending on whether the value exceeds the maximum value or is lower than the minimum value).

Item	Description
Minimum Value	The minimum value for the gauge scale of measurement. By default this is 0.
Maximum Value	The maximum value for the gauge scale of measurement. By default this is 100.
Minor Unit	Interval at which tick marks should display on the gauge to indicate values along the measurement scale. By default this is 5.
Major Unit	Interval at which number labels should display on the gauge to indicate values along the measurement scale. By default this is 20. The minimum value and the maximum value are always labeled on the gauge.
Label Format	Specifies the format for the major unit labels: **Number** (default), **Currency**, or **Percent**.
Label Decimals	Specifies the number of decimals to display on the major unit labels. By default, no decimal places are shown (0).
Labels Are Outside Gauge	Specifies the placement of the gauge labels and unit markers, either inside the gauge (in the area where the pointer sweeps) or outside of the gauge. By default this option is not selected, which means labels are placed inside the gauge. The example screenshot before this table shows the default behavior of labels inside the gauge.
Orientation	Specifies the orientation of the gauge: **Vertical** (default) or **Horizontal**.
Pointer Color	The color of the pointer. If left blank, the pointer color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.

Range settings

You can define up to three ranges for the gauge. Ranges are defined by a starting and ending value, and a color to shade that range. If you do not want to use a particular range, leave the settings for that range blank.

If you want the ranges to be continuous, then the **To** value of one range and the **From** value of the next range should be the same number. For example, if range one is from 0 to 20, then the from value for range two should be 20.

Range colors can be inherited from the style or skin (in that order), or colors can be manually specified. By default, all platform skins are set to use green, yellow, and red.

Item	Description
Range 1 From	The starting value of the range.
Range 1 To	The ending value of the range.
Range 1 Color	The color for the range.
Range 2 From	The starting value of the range.
Range 2 To	The ending value of the range.
Range 2 Color	The color for the range.
Range 3 From	The starting value of the range.
Range 3 To	The ending value of the range.
Range 3 Color	The color for the range.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Even though Pointer Color and the range colors can be affected by styles, these properties are exposed as component behavior properties because they are unique to the gauge component type. Also, the Axiom Software platform does not currently provide any styles specifically designed for gauge components.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Scatter Chart component

The Scatter Chart component illustrates data with two variables using a collection of points, where one variable determines a data point's position on the horizontal axis, and a second variable determines the data point's position on the vertical axis.

Scatter charts are part of the ScatterChart family, which includes scatter line and bubble charts. All of these charts use the same data source type (ScatterChart) and have the same basic component properties.

Defining a scatter chart is a two-part process that requires the following:

- Creation of a ScatterChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Scatter Chart component on the Axiom form canvas.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Scatter Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

`[ScatterChart;DataSourceName]`

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

`[Series]`

Each row flagged with this tag defines a data point to be displayed in the chart. Multiple rows can belong to the same series, depending on the name entered in the `[SeriesName]` column.

Column tags

`[XValue]`

This column contains the values to determine the x-axis position (horizontal) of each data point.

`[YValue]`

This column contains the values to determine the y-axis position (vertical) of each data point.

[Label]

Optional. This column contains the name of each individual data point in the chart. By default, the label will display in a tooltip when the user hovers over the data point. If labels are enabled for the chart, then the label will also display next to the data point within the chart. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[Color]

Optional. This column specifies the color assignment for each series or each data point. If omitted, then colors will be dynamically determined based on the style, theme, or skin (in that order). See Specifying chart colors. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

If you are specifying colors and you want the color to apply to the entire series, then only use one color entry per series (leaving the rest blank), or repeat the same color entry on all items of the series.

Column tags (optional, series-wide)

All of the following tags are optional and apply to the entire series, not just the current data point. If you use any of these series-wide tags, you should make sure that each entry in the tag is the same for all data points that belong to the same series. If different entries are found within the same series, the first entry is used.

[SeriesName]

This column contains the names of each series in the chart. Multiple data points in the chart can belong to the same series by entering the same series name in this column. If this column is omitted, then all data points in the chart belong to a single unnamed series.

[Kind]

This column indicates the kind of each series, either Scatter, ScatterLine, or Bubble. If omitted, then all series in the chart will use the Default Series Kind defined in the component settings.

[VisibleinLegend]

This column indicates whether a series is shown in the chart legend, if the legend is enabled. If omitted, all series are shown. Indicate True or False. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[LineWidth]

For Scatter Line series only. This optional column indicates the width of the scatter line, in pixels. If omitted, then the line will be 1 pixel. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[MarkerSize]

This optional column indicates the size of the marker for the series. Enter 1-10 with 1 as the smallest marker size and 10 as the largest size. If omitted, then the marker size is 6. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[MarkerType]

This optional column indicates the marker type for the series. Enter Circle, Triangle, or Square. If omitted, Circle is used. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

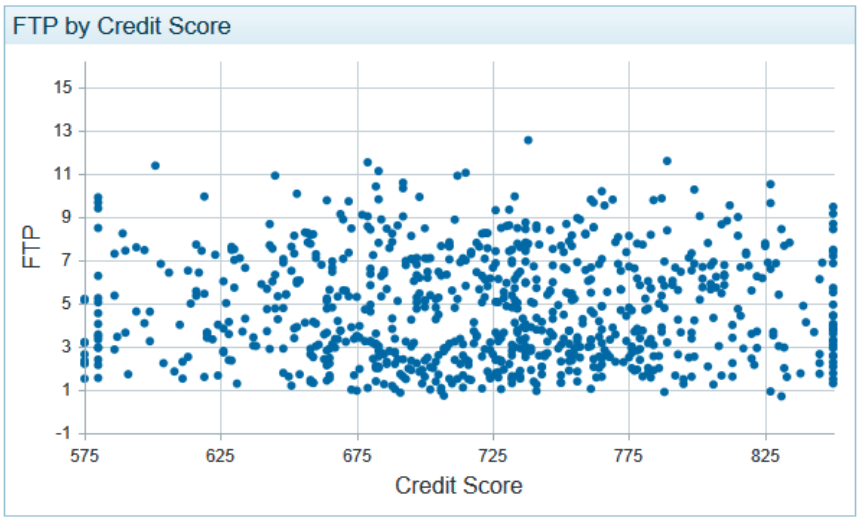
The following example shows simple data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data. This example shows only the columns that are added by the Data Source Wizard; if you want to use the other columns then you must manually add them.

	A	B	C	D	E	F
4						
5		[ScatterChart;ScatterBasic]	[Kind]	[SeriesName]	[Xvalue]	[YValue]
6		[Series]	Scatter	Customers	777	6.07037439
7		[Series]	Scatter	Customers	724	8.57414459
8		[Series]	Scatter	Customers	720	7.14711168
9		[Series]	Scatter	Customers	782	6.11624277
10		[Series]	Scatter	Customers	590	3.71379548
11		[Series]	Scatter	Customers	730	8.57008491
12		[Series]	Scatter	Customers	792	7.56379288
13		[Series]	Scatter	Customers	791	4.28439929
14		[Series]	Scatter	Customers	723	8.70511602

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Scatter Chart**. If the data already exists in the sheet, you can first highlight the labels and data and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the

row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

The resulting chart would appear as follows:



NOTE: This chart uses the default marker shape and size. The labels for the X-Axis and Y-Axis ("FTP" and "Credit Score") are defined in the component properties, not in the data source.

Component properties

You can define the following properties for a Scatter Chart component:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.

Item	Description
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. Select either **Bubble**, **Scatter**, or **Scatter Line**. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Bubble Chart on the canvas, then the default is automatically set to Bubble.
Show Labels	Specifies whether labels will display next to each data point in the chart. Labels are defined in the optional [Label] column within the data source. If your chart has many data points, you may want to disable this setting to avoid clutter in the chart. The labels will still display in tooltips when the user hovers over the data point, if labels are defined in the data source.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Y-Axis Label	Optional. Enter a label for the Y-axis (vertical). This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Y-Axis Format	Optional. Specify the format for the Y-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Y-Axis Decimals	Optional. Specify how many decimal places to show on the Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Y-Axis Min Y-Axis Max	Optional. Specify the maximum value and the minimum value for the Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
X-Axis Label	Optional. Enter a label for the X-axis (horizontal). This will display next to the X-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
X-Axis Format	Optional. Specify the format for the X-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
X-Axis Decimals	Optional. Specify how many decimal places to show on the X-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
X-Axis Min X-Axis Max	Optional. Specify the maximum value and the minimum value for the X-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
X-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the X-axis. By default, no scale is applied. Enter a number to scale all X-axis numbers by that value. The X-axis numbers will be divided by the specified value. For example, if an X-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the X-axis scale, not 35.
Bubble Units Bubble Size Format Bubble Size Decimals	These items do not apply to Scatter Charts or Scatter Line Charts. However, they still display in the component properties in case you change the series kind for one or more series to Bubble. For more information on these properties (and on the additional data source tags for Bubble charts), see Bubble Chart component.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the ScatterChart family. Only the generic styles are available.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Scatter Line component

The Scatter Line Chart component illustrates data with two variables using a linked collection of points, where one variable determines a data point's position on the horizontal axis, and a second variable determines the data point's position on the vertical axis.

Scatter line charts are part of the ScatterChart family, which includes scatter and bubble charts. All of these charts use the same data source type (ScatterChart) and have the same basic component properties.

Defining a scatter line chart is a two-part process that requires the following:

- Creation of a ScatterChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Scatter Line Chart component on the Axiom form canvas.

Scatter Line Charts are most commonly implemented as combination scatter point and scatter line charts. In this case, you can start with either the Scatter Line Chart or Scatter Chart component and then modify the data source to use both types of series. For more information, see Scatter Chart component .

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Scatter Line Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[ScatterChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a data point to be displayed in the chart. Multiple rows can belong to the same series, depending on the name entered in the `[SeriesName]` column.

Column tags

[XValue]

This column contains the values to determine the x-axis position (horizontal) of each data point.

[YValue]

This column contains the values to determine the y-axis position (vertical) of each data point.

[Label]

Optional. This column contains the name of each individual data point in the chart. By default, the label will display in a tooltip when the user hovers over the data point. If labels are enabled for the chart, then the label will also display next to the data point within the chart. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[Color]

Optional. This column specifies the color assignment for each series or each data point. If omitted, then colors will be dynamically determined based on the style, theme, or skin (in that order). See Specifying chart colors. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

If you are specifying colors and you want the color to apply to the entire series, then only use one color entry per series (leaving the rest blank), or repeat the same color entry on all items of the series.

Column tags (optional, series-wide)

All of the following tags are optional and apply to the entire series, not just the current data point. If you use any of these series-wide tags, you should make sure that each entry in the tag is the same for all data points that belong to the same series. If different entries are found within the same series, the first entry is used.

[SeriesName]

This column contains the names of each series in the chart. Multiple data points in the chart can belong to the same series by entering the same series name in this column. If this column is omitted, then all data points in the chart belong to a single unnamed series.

[Kind]

This column indicates the kind of each series, either Scatter, ScatterLine, or Bubble. If omitted, then all series in the chart will use the Default Series Kind defined in the component settings.

[VisibleinLegend]

This column indicates whether a series is shown in the chart legend, if the legend is enabled. If omitted, all series are shown. Indicate True or False. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[LineWidth]

For Scatter Line series only. This optional column indicates the width of the scatter line, in pixels. If omitted, then the line will be 1 pixel. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[MarkerSize]

This optional column indicates the size of the marker for the series. Enter 1-10 with 1 as the smallest marker size and 10 as the largest size. If omitted, then the marker size is 6. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

[MarkerType]

This optional column indicates the marker type for the series. Enter Circle, Triangle, or Square. If omitted, Circle is used. This tag is not added by the data source wizard; you must manually add it to the data source if you need it.

NOTES:

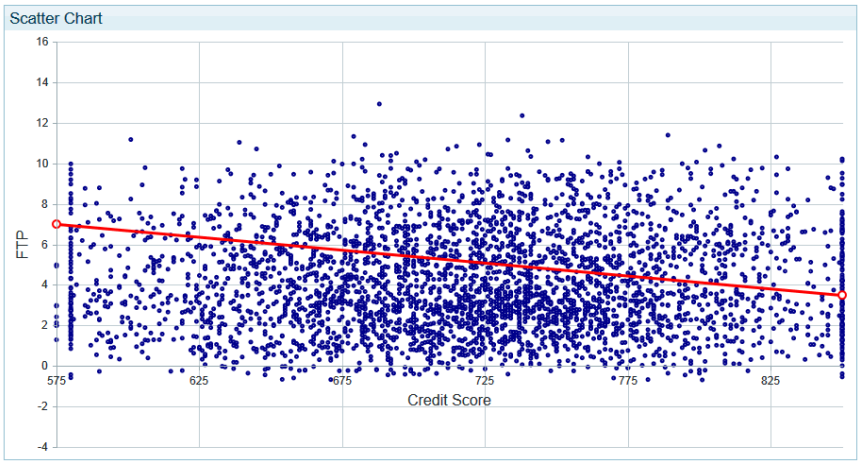
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

The following example shows simple data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I
1									
2									
3		[ScatterChart;ScatterCombo]	[Kind]	[SeriesName]	[LineWidth]	[MarkerSize]	[Color]	[Xvalue]	[YValue]
4		[Series]	ScatterLine	Trend	3		Red	575	7.00000000
5		[Series]	ScatterLine	Trend	3		Red	850	3.50000000
6		[Series]	Scatter	Customers		3	DarkBlue	696	3.25622555
7		[Series]	Scatter	Customers		3	DarkBlue	745	5.12440196
8		[Series]	Scatter	Customers		3	DarkBlue	580	1.97421227
9		[Series]	Scatter	Customers		3	DarkBlue	744	2.81894857
10		[Series]	Scatter	Customers		3	DarkBlue	783	1.22620801
11		[Series]	Scatter	Customers		3	DarkBlue	662	-0.21592974

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Scatter Line**. If the data already exists in the sheet, you can first highlight the labels and data and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

In this case the data is a combination scatter and scatter line chart, which is a common combination. The resulting chart would appear as follows:



Component properties

You can define the following properties for a Scatter Line Chart:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.

Item	Description
Default Series Kind	Specifies the default kind for series in the chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. Select either **Bubble**, **Scatter**, or **Scatter Line**. When you place a chart component on the canvas, the Default Series Kind is automatically set based on the type of chart you used. For example, if you drag and drop a Bubble Chart on the canvas, then the default is automatically set to Bubble.
Show Labels	Specifies whether labels will display next to each data point in the chart. Labels are defined in the optional [Label] column within the data source. If your chart has many data points, you may want to disable this setting to avoid clutter in the chart. The labels will still display in tooltips when the user hovers over the data point, if labels are defined in the data source.
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Y-Axis Label	Optional. Enter a label for the Y-axis (vertical). This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Y-Axis Format	Optional. Specify the format for the Y-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Y-Axis Decimals	Optional. Specify how many decimal places to show on the Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Y-Axis Min Y-Axis Max	Optional. Specify the maximum value and the minimum value for the Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
X-Axis Label	Optional. Enter a label for the X-axis (horizontal). This will display next to the X-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
X-Axis Format	Optional. Specify the format for the X-axis: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
X-Axis Decimals	Optional. Specify how many decimal places to show on the X-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
X-Axis Min X-Axis Max	Optional. Specify the maximum value and the minimum value for the X-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
X-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the X-axis. By default, no scale is applied. Enter a number to scale all X-axis numbers by that value. The X-axis numbers will be divided by the specified value. For example, if an X-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the X-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the X-axis scale, not 35.
Bubble Units Bubble Size Format Bubble Size Decimals	These items do not apply to Scatter Charts or Scatter Line Charts. However, they still display in the component properties in case you change the series kind for one or more series to Bubble. For more information on these properties (and on the additional data source tags for Bubble charts), see Bubble Chart component.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the ScatterChart family. Only the generic styles are available.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Sparkline component

The Sparkline component for Axiom forms can be used to show trends at-a-glance, in a simple format. Unlike full-sized charts which are intended to stand alone and convey detailed data, sparklines are intended to be presented along with text and data, to illustrate what is being conveyed.

The only content of a sparkline chart is the data plotting element (for example, the chart line, or the chart columns)—no axis values or labels of any kind display on the chart. The goal of the sparkline chart is to convey the essence of the data, not the specific details—for example, is the data generally trending up, down, or fluctuating? However, users can hover their cursor over distinct points of the sparkline chart to see the specific value for that point in a tooltip.

Defining a sparkline chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Sparkline component on the Axiom form canvas.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Sparkline components use the same data source tags as Bar Chart, Column Chart, Line Chart, and Area Chart components—meaning the XYChart data source. Any XYChart series kind can be displayed in the sparkline chart.

Note the following exceptions when using an XYChart data source for a sparkline chart:

- Sparkline charts do not display axis labels or legends, so the `[XValueName]` row and the `[VisibleinLegend]` column are not applicable. The `[SeriesName]` column must still be used to identify series in the data source, but the series names do not display in the chart.
- Sparkline charts only support one Y-axis scale, so the `[Axis]` column is not applicable. If `[Axis]` is present in the data source and any series are assigned to the Secondary axis, those series will not display in the sparkline chart.

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each sparkline chart uses a single series in the data source.

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. The name identifies this series so that it can be assigned to a chart component.

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

[Kind]

Specifies the kind of each series in the chart: `Area`, `Bar`, `Column`, `Line`, or `Waterfall`. If omitted, then all series in the chart will use the Default Series Kind as defined in the component settings. If a data source contains multiple kinds of series then it is known as a combination chart (for example, one or more column series combined with a line series).

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

[LineStyle]

Optional. Specifies the style of the line as one of the following. If omitted, the Normal style is used. Only applies to Line series.

- **None**: No line is displayed; only markers are shown to represent the data points. `[ShowMarkers]` must be enabled or else the series will not display at all. This option is primarily intended for use in combination charts—for example, multiple bar series combined with a marker-only line series.
- **Normal**: A straight line is drawn from point to point.
- **Smooth**: A curved line is drawn from point to point.
- **Step**: The line "steps" from one point to another. The lines between points are flat, with a vertical line up or down to indicate the differential at each point.

[DashType]

Optional. Specifies the type of dash as one of the following. If omitted, the Solid style is used. Only applies to Line series.

- **Dash**: The line is drawn in dashes. The length of the dash is fixed and cannot be configured.
- **DashDot**: The line is drawn as a dash-dot-dash repeating series.
- **Dot**: The line is drawn in dots. The size of the dot is fixed and cannot be configured.
- **Solid**: The line is drawn as a solid line.

[PlotNullValues]

Optional. Specifies whether null values are plotted on the line (True/False). Only applies to Line series.

If omitted or False, then null values will result in a gap between line segments. If True, then the missing value will be interpolated, so that the line will continue from the last plotted point to the next plotted point.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2		[XYChart;Rates]	[Kind]	[SeriesNa	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[Series]	Line	Rate	3.53	3.51	3.57	3.4	3.81	4.46	4.31	4.51	4.32	4.1	4.29	4.48

The resulting chart would appear as follows (where the chart title is a separate Label component):

When using **Create Axiom Form Data Source** on the right-click menu, there is no separate option for Sparkline; instead you should select Line Chart, Column Chart, etc., based on what type of chart you want to display in the sparkline chart. Traditionally sparklines are line charts, but you can use any of the chart kinds supported by the XYChart data source. To see an example data source, see the topic for the type of chart that you want to display as a sparkline. For example, if you want to display a Line Chart as a sparkline, see the topic for *Line Chart component*.

When configuring the settings for the Sparkline component, you can select not only which data source to use, but also which series from that data source to use. Therefore you may have multiple Sparkline components that use the same data source, but each sparkline chart displays a different series in the data source. In this example the data source has only one series, but it could have multiple series with multiple components referencing the same data source.

Component properties

You can define the following properties for a Sparkline component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the sparkline chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.
Data Source Access	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Selected Series	Optional. The specific series of the data source to show in the chart. The drop-down list shows all currently visible series names in the data source. You can select one of these names or type in a name (in case you haven't defined the series name yet, or if it is currently "hidden" via a formula). This setting allows you to define multiple series within a single data source, and then display each series within a separate sparkline chart. If you do not specify a series, then all series in the data source will display in the chart. Keep in mind that if the data source contains more than a small handful of series, the sparkline chart will likely be unreadable.

Item	Description
Default Series Kind	Specifies the default kind for series in the sparkline chart, to be used if the Kind column is omitted from the data source, or if an entry in the column is blank. Select either **Bar**, **Column**, **Line**, or **Area**. The default kind for sparklines is Line, since that is the traditional chart type of sparklines.
Display Tooltip	Specifies whether tooltips display on the chart when a user hovers the cursor over it. By default this is enabled. A tooltip is available for each individual value in the series, displaying the series name and value.
Tooltip Format	Optional. Specifies the format for values displayed in the sparkline tooltip: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • If you select Percent, the numbers in the series should be formatted as percent in the spreadsheet, so that when a user hovers over the value it will display in percent format.
Tooltip Decimals	Optional. Specifies how many decimal places to show in the sparkline tooltip. By default, no decimal places are shown (0).
URL	Optional. The URL to launch when a user clicks on the component. The URL must use full HTTP syntax—meaning, use `HTTP://www.axiomepm.com`, not `www.axiomepm.com`. For example, you might use the URL to open another Axiom form with more detailed information about the data in the sparkline chart. For more information about using a URL to open another Axiom form, see Generating a URL to an Axiom file or an Axiom form.
Use New Window	If a URL is defined, specifies whether the link is opened in a new window. By default this is enabled, which means the link is opened in a new window. Disable this option if you want the link to open within the same window (replacing the current Axiom form).

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for Sparkline components. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Sparkline alternatives

Axiom forms often support several different ways of performing the same task, to provide a broad range of display options and user interface behavior. Depending on your form design, you may want to consider the following alternatives:

- The Sparkline content tag can be used in Formatted Grid components to present sparklines within a grid. In many cases this approach may be preferred to place the sparkline in context with its associated data. For more information, see Using sparklines in Formatted Grids.

Waterfall Chart component

Waterfall Chart components illustrate the cumulative effect of positive and negative contributions to a starting value. Waterfall charts may also be known as "flying bricks charts," due to the presentation of the values in the chart.

Waterfall charts are part of the XYChart family, which includes charts such as column, line, and area charts. All of these charts use the same data source type (XYChart) and have the same basic component properties.

Defining a waterfall chart is a two-part process that requires the following:

- Creation of an XYChart data source in the spreadsheet to define the data to display in the chart.
- Placement and configuration of a Waterfall Chart component on the Axiom form canvas.

Waterfall charts can also support interactivity, to change the contents of the Axiom form based on the currently selected data point in the chart.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Data source tags

Waterfall Chart components must have a defined data source within the source file to indicate the data for the chart. The tags for the data source are as follows:

Primary tag

[XYChart;*DataSourceName*]

The DataSourceName identifies this data source so that it can be assigned to a chart component. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save.

The placement of this primary tag defines the control column and the control row for the data source.

- All column tags must be placed in this row, to the right of the tag.
- All row tags must be placed in this column, below the tag.

Row tags

[Series]

Each row flagged with this tag defines a series of data to be displayed in the chart. Each series will use a different color. Typically, Waterfall charts only use one series.

[XValueName]

This row contains the names of each XValue column in the chart. These names will display along the primary axis of the chart (the X-axis for most charts; the Y-axis for bar charts).

Column tags

The data source wizard only adds the `[SeriesName]`, `[XValue]`, and `[Kind]` columns. If you want to use any of the other columns, you must manually add them to the data source.

[SeriesName]

Defines the name of each series in the chart. These names will be displayed in the chart legend, if the chart is configured to show a legend (as defined in the component settings).

[XValue]

Each column of data to be displayed in the chart must be marked with an XValue tag.

For waterfall series, the first XValue column determines the starting value. Each subsequent XValue column must then contain a positive or negative value to be added to or subtracted from the cumulative running total. For example, if the starting value is 250 and you want to show that the next value is increased by 45 (reaching a total value of 295), then the second XValue column should contain 45 instead of 295. See the data source example at the end of the section for more information.

Additionally, you can use the following special keywords in XValue columns, to display total bars in the chart:

- `RunningTotal`: Displays the sum of all items since the last RunningTotal point. For example, if the previous three columns were the values for Jan-March, you could use a RunningTotal to display the total change for Q1.
- `Total`: Displays the sum of all previous items. For example, this could be the last XValue item in the chart, to display the total of all previous values.

[Kind]

For Waterfall charts, the kind is always the Default Series Kind as specified in the component properties. Specifying a different kind here has no effect. You can leave the Kind column blank, or you can set it to match the Default Series Kind.

[Color]

Optional. Specifies the color assignment for each series. If omitted, then colors will be dynamically determined based on the style or skin (in that order). See Specifying chart colors.

For Waterfall Charts, you can use additional optional columns to specify different colors for total columns and running total columns.

[RunningTotalColor]

Optional. Specifies the color assignment for columns that use the `RunningTotal` keyword. If omitted, these columns will use the same color as regular value columns.

[TotalColor]

Optional. Specifies the color assignment for columns that use the Total keyword. If omitted, these columns will use the same color as regular value columns.

[Axis]

Optional. Specifies the Y-axis scale for each series. This column is only required if the chart has both a primary and secondary Y-axis. If omitted, the primary Y-axis scale is assumed. See Using two Y-axis scales with combination XYCharts.

[VisibleinLegend]

Optional. Specifies whether a particular series is shown in the chart legend (True/False). If omitted, all series are shown. This setting only applies if the chart is configured to show a legend (as defined in the component settings).

NOTES:

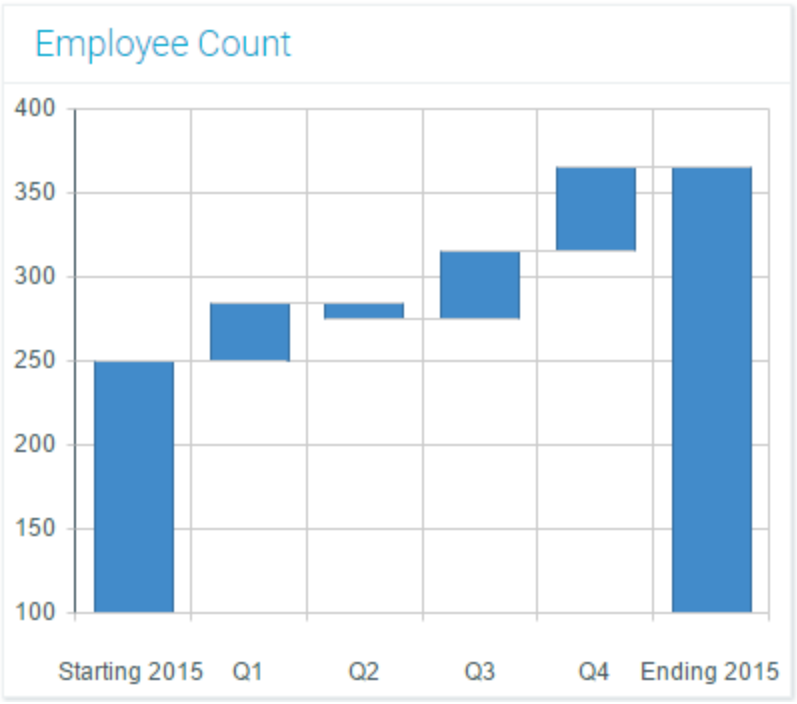
- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.
- Negative numbers in a data source must use the minus symbol or parentheses to indicate the negative value. Alternative negative formats such as red number text are not recognized and will display as positive values in the chart.

The following example shows simple employee count data flagged in a sheet. In real implementations this data would most likely be generated by an Axiom query or Axiom functions; here the data is simply typed in order to show the placement of the tags to the data.

	A	B	C	D	E	F	G	H	I	J
6										
7		[XYChart;EmployeeChange]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
8		[XValueName]		2015	Starting 2015	Q1	Q2	Q3	Q4	Ending 2015
9		[Series]	Waterfall	Employees	250	35	-10	40	50	total

To use the Data Source Wizard to add the tags to a sheet, right-click in the cell where you want to start the data source and then select **Create Axiom Form Data Source > Waterfall Chart**. If the data already exists in the sheet, you can first highlight the labels and the values (in the example above, you would highlight D3:I4) and then use the wizard. Axiom Software will add the tags as displayed in the example above, including adding the `[Kind]` column. The cells in the row above and the column to the left of the highlighted area must be blank in order for Axiom to place the tags in sheet.

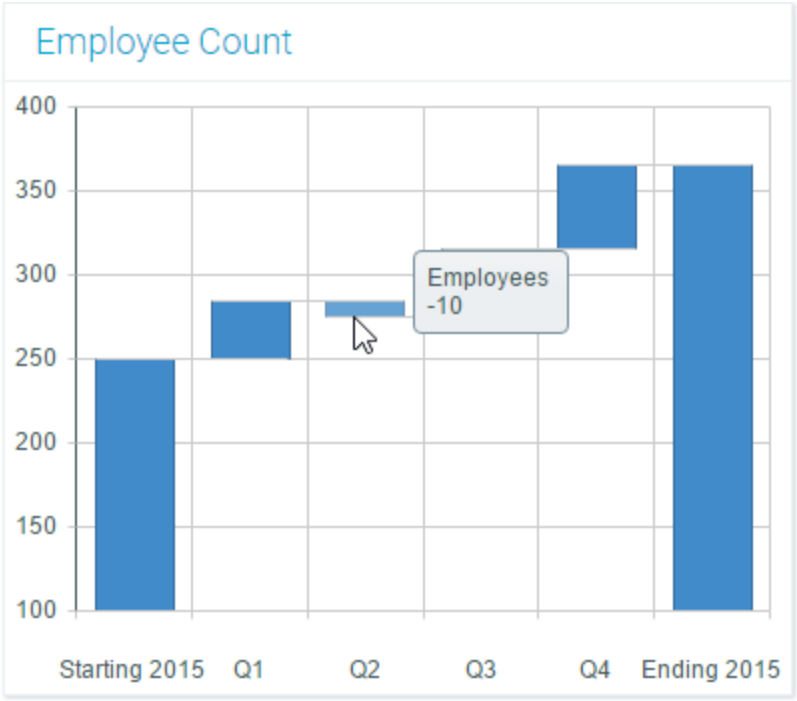
The resulting chart would appear as follows:



This chart shows the number of employees at the start of the year, followed by the change in employee count for each subsequent quarter. For example:

- In Q1 this company grew by 35 employees, so the Q1 XValue column in the data source contains the number 35. This Q1 number is shown in the chart as the difference between the starting value of 250 and the new total value of 285. Notice the Q1 column in the chart starts at the 250 mark and extends to the new total value, instead of starting at the baseline X-axis. This is to show the differential in relation to the previous value.
- In Q2 this company shrank by 10 employees, so the Q2 XValue column in the data source contains the number -10. This Q2 number is shown in the chart as the difference between the prior total value of 285 and the new total value of 275.
- The final column in the chart shows the total of all previous values, by using the special keyword of `Total` as the XValue.

The form user can see the exact XValue amount of each column by hovering their cursor over the column in the chart.



Component properties

You can define the following properties for a Waterfall Chart component:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Data Source	The data source for the chart. You must have defined the data source within the file using the appropriate tags in order to select it for the chart. **NOTE:** In the Form Control Sheet, the selected data source is written as *SheetName*!*DataSourceName*. The sheet name is the sheet where the selected data source is located. If a data source is already specified and you want to locate it within the file, click the **Show location** button to the right of the drop-down list. This will move your cursor to the associated data source tag in the file.

Item	Description
Data Source Load	Specifies the loading behavior of the component: • **Inline** (default): The component properties and data are both loaded when the form is loaded. This behavior causes the overall form load to take longer, because the component data must be loaded before any of the form can display on the web page. However, once the form does load, the component is fully rendered. • **Asynchronous**: When the form is loaded, the component "shell" is loaded and rendered on the web page without the underlying data from the data source. This behavior speeds up the initial load of the form, because it does not have to wait for the component data to load. Once the form is rendered, a second pass is performed to load the component data. A loading spinner displays within the component "placeholder" until the data has finished loading.
Selected Label Selected Series	The currently selected data point in the chart. This is identified by the corresponding label for the data point (the XValueName) and the Series that the data point belongs to. These settings are only used if the chart is configured to support interactivity. These settings serve two purposes: • They specify the initially selected data point of the chart, when the user first opens the form. You can leave the settings blank to have no initial selection, or you can enter an XValueName from the data source into the Selected Label field, and the corresponding Series name into the Selected Series field. The initial selection is not highlighted in the form, but it will determine the initial state of any other components that reference these settings. • When a user views the form and selects a data point in the chart, the XValueName and Series name of the selected point will be submitted back to the source file and placed in these cells on the Form Control Sheet. Formulas can reference these cells in order to dynamically change the form based on the currently selected data point in the chart.
Auto Submit	Specifies whether the Axiom form is automatically refreshed when a user selects a data point in the chart. By default, this is disabled. You should leave this option disabled if you have not set up your chart to support interactivity; otherwise the Axiom form will refresh unnecessarily if the user clicks on data points in the chart. If enabled, then the form automatically refreshes when the user selects a data point in the chart. It is recommended to enable this option if the chart is set up to support interactivity, so that the user gets immediate feedback on their selection.

Item	Description
Title Text	The title text for the chart. This text displays in the title bar of the chart panel within the Axiom form, if the title bar is enabled. If the title bar is not enabled, then the text displays centered over the top of the chart. **NOTE:** The font type / size / weight / style of the title text are dependent on the style or skin and cannot be changed.
Show Title Bar	Specifies whether the title bar is visible. By default this option is enabled, which means that the component will display in a bordered box with a title bar across the top. The defined title text displays within the bar. The formatting of the title bar and its border are determined by the skin specified for the form. If disabled, then the title bar and its border will not display on the component. If the title bar is enabled and the component also has a separately defined border (either via a style or by using the formatting overrides in the advanced component settings), then both borders will display on the component. In some cases this effect may be desired; in other cases one of the borders should be disabled.
Legend	The location of the chart legend. You can specify **None** for no legend, or specify a location such as **Top**, **Bottom**, **Right**, or **Left**. If you are using a legend, and you want to omit a series from displaying in the legend, you can use the optional column `[VisibleinLegend]` for the data source. Legends not only identify each series in the chart, they can also be used to dynamically show and hide series in the chart. Users can click on a series name in the legend to toggle that series hidden and visible.
Default Series Kind	Specifies the type of waterfall series to show in the chart. Select either **Waterfall** or **Horizontal Waterfall**. This selection takes precedence over any kind entered into the Kind column of the data source.
Composition Kind	This option does not apply to waterfall series and will be ignored. Waterfall charts typically only have one series. If multiple waterfall series are present, they will be shown side-by-side but the connecting lines for each column may not display as expected.
Area Series Opacity	Specifies the opacity of area series within the chart: • **Opaque** (default): Area series are opaque. • **Translucent**: Area series are translucent. This is typically selected if the **Composition Kind** of the chart is set to **Side by Side**, so that you can see all areas in the chart. This setting is ignored for all other series kinds.

Item	Description
Show Grid Lines	Specifies whether gridlines display on the chart. By default, this is enabled.
Show Axes	Specifies whether the axis labels display on the chart. By default, this is enabled. Disabling this option hides the XValueNames defined in the data source, and the scale values for both axes. **NOTE:** If an optional Y-axis label is defined, it will display regardless of this setting.
Name Rotation	The degree of rotation for the chart names (the XValueNames from the data source). By default this is blank, which means that the names are not rotated. To rotate the names, enter a value from -360 to 360. The purpose of this setting is to allow displaying longer names as vertical or slanted. For example, a value of -45 displays the name as slanted upward, whereas a value of 45 displays the name as slanted downward.  *-45 degree name rotation*  *45 degree name rotation*
Primary Y-Axis Label	Optional. The label for the primary Y-axis. This will display next to the Y-axis scale. For example, if the scale is dollars in millions, you can define a label of "Dollars" or "Dollars in Millions".
Primary Y-Axis Format	Optional. Specifies the format for the primary Y-axis values: **Number** (default), **Currency**, or **Percent**. **NOTES:** • If you select Currency, the currency symbol is determined by your operating system locale. • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.
Primary Y-Axis Decimals	Optional. Specifies how many decimal places to show on the primary Y-axis labels. By default, no decimal places are shown (0). **NOTE:** This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet.

Item	Description
Primary Y-Axis Min Primary Y-Axis Max	Optional. Specifies the maximum value and the minimum value for the primary Y-axis labels. If omitted, the maximum and minimum values will be determined by the values in the series. For example, you might use this option if you want to show a full percent scale from 0% to 100%, even though the minimum and maximum values in the series are 25% and 83%. **NOTE:** If the series format is percent, the minimum and maximum values should be entered in the decimal equivalent. For example, enter 1 if you want the maximum to be 100%.
Primary Y-Axis Scale	Optional. Specifies a scaling property for the numbers displayed along the Y-axis. By default, no scale is applied. Enter a number to scale all Y-axis numbers by that value. The Y-axis numbers will be divided by the specified value. For example, if a Y-axis value is 25,000,000 and the scale is 1000, the value will be displayed as 25,000. If the scale is 1000000, then the value will be displayed as 25. **NOTES:** • This setting only impacts the Y-axis numbers. The actual chart values (shown in tooltips) will continue to display as they are formatted in the spreadsheet. • If a scale property is defined, the Min and Max values should reflect the original values before scaling is applied, not the scaled values. For example, enter 35,000,000 if you want that to be the top value on the Y-axis scale, not 35.
Use Secondary Y-Axis	Select this option if you want to create a chart with two different Y-axis scales. If this check box is selected, then another series of Y-Axis settings will display for the Secondary Y-Axis. These settings work the same way as the settings for the Primary Y-Axis. Typically, multiple Y-axis scales are only used with combination charts, meaning charts with two types of series. For more information, see Creating combination charts.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Currently, the Axiom Software platform does not provide any styles specifically designed for charts in the XYChart family. Only the generic styles are available.

NOTE: The colors used in the chart are determined by the data source. If colors are not specified in the data source, then they are determined by the style, theme, or skin (in that order).

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

▶ Interactive behavior

The Waterfall Chart component can be set up to allow the user to select a column in the chart. The selected column is submitted back to the source file, and written to the **Selected Label** and **Selected Series** settings on the Form Control Sheet, using the XValueName and the corresponding Series name of the selected column.

If you want the Axiom form to respond to the currently selected column, then you must set up the file so that another component references the selected label and/or series and changes based on it. For more information on setting up interactive components for an Axiom form, see Using interactive components in an Axiom form.

Chart interactivity is intended to support chart drilling based on the currently selected item. For example the user may want to see more detail about the data that makes up a particular column in the chart.

Example

The Axiom form could contain a waterfall chart that shows the change in employee count by month. If you want users to be able to see the details about the data in any particular month, you could set up a second chart that references the selected label and series of the first chart. For example, if the user selects the column for February in the first chart, the second chart will be updated to show detailed employee data for February.

Creating combination charts

Certain chart types can be combined into a single chart, so that some data series in the chart use one chart type, and other data series use a different chart type. A common example of a combination chart is a column chart that also contains a line series, where the line indicates a target or an average to compare to the column data.

Only chart types in the same chart family can be combined. For example, columns and lines can be combined in a single chart because they both belong to the XYChart family.

Creating a combination chart

To create a combination chart, you must use the `[Kind]` column in the data source to indicate which chart type you want each series to be. For example, if you want to create a combination column and line chart, you could do this as follows:

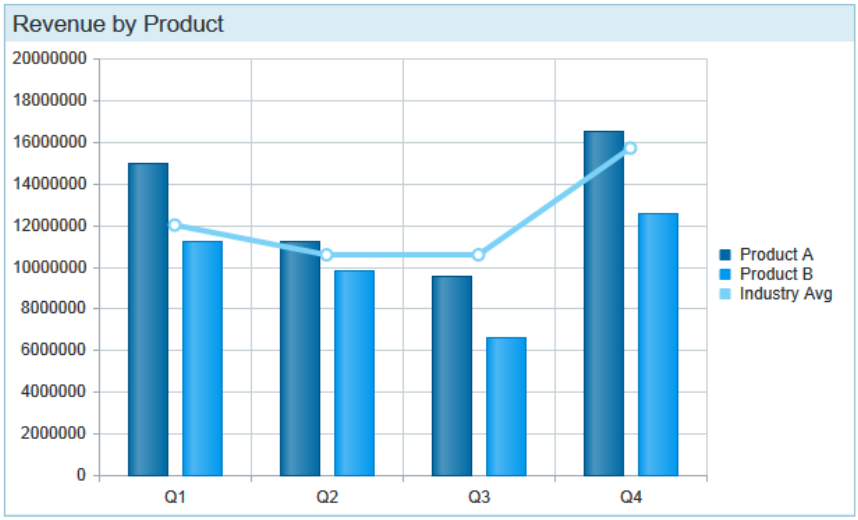
- Set up your data in the sheet, and then use **Create Axiom Form Data Source > Column Chart** to insert the data source tags. This means that any existing data in the data source will be configured to use `Column` as the chart type.
- Modify the `[Kind]` column to put `Line` on the series you want to display as a line.

	A	B	C	D	E	F	G	H
1								
2		[XYChart;ComboSource]	[Kind]	[SeriesName]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]			Q1	Q2	Q3	Q4
4		[Series]	Column	Product A	15000000	11235900	9564852	16523987
5		[Series]	Column	Product B	11258741	9852741	6589321	12569874
6		[Series]	Line	Industry Avg	12000000	10569987	10569963	15693825
7								

- Drag and drop a Column Chart component onto the canvas, and configure that component to use this data source.

You could also insert the data source tags first, and then configure the data and the `[Kind]` column as needed for each series. It does not matter which type of data source you insert or which type of chart component you place on the canvas, as long as they belong to the appropriate chart family (XYChart in this example). The data source type and component type simply determine the default series kind used by the chart. However, when the chart is rendered, the ultimate determiner of the chart type is the series kind in the data source.

In this example, the resulting combination XYChart would appear as follows. The chart has two column series and one line series, as determined by the data source.

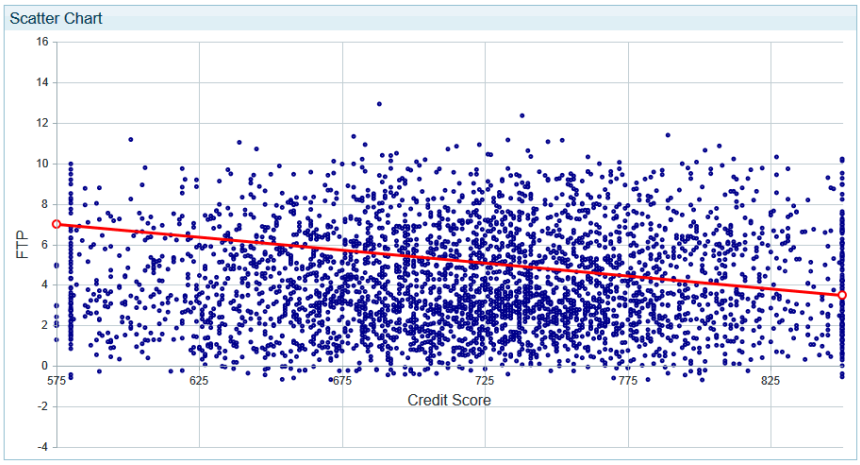


In this example, the line series is using the same scale as the column series. However, it is also possible to use a second Y-axis scale for the line series. For more information, see Using two Y-axis scales with combination XYCharts.

The same basic approach applies to ScatterChart combinations. The following screenshot shows an example ScatterChart data source with two series types in the `[Kind]` column. The majority of the series are Scatter, with two ScatterLine points to display a trend line.

	A	B	C	D	E	F	G	H	I
1									
2									
3		[ScatterChart;ScatterCombo]	[Kind]	[SeriesName]	[LineWidth]	[MarkerSize]	[Color]	[Xvalue]	[YValue]
4		[Series]	ScatterLine	Trend	3		Red	575	7.00000000
5		[Series]	ScatterLine	Trend	3		Red	850	3.50000000
6		[Series]	Scatter	Customers		3	DarkBlue	696	3.25622555
7		[Series]	Scatter	Customers		3	DarkBlue	745	5.12440196
8		[Series]	Scatter	Customers		3	DarkBlue	580	1.97421227
9		[Series]	Scatter	Customers		3	DarkBlue	744	2.81894857
10		[Series]	Scatter	Customers		3	DarkBlue	783	1.22620801
11		[Series]	Scatter	Customers		3	DarkBlue	662	-0.21592974

The resulting combination ScatterChart would appear as follows:



Supported combinations and design considerations

XYChart types can be combined together in one chart, and ScatterChart types can be combined together in one chart. The following chart combinations are the most commonly used:

- Column and Line (XYChart)
- Bar and Line (XYChart)
- ScatterLine and Scatter (ScatterChart)

Other combinations may not be valid or may not render as expected. Note the following design considerations for each series kind:

Series Kind	Design Considerations for Combination Charts
Area	Although it is possible to combine area series with other series kinds, the results may not be as you expect. One issue to consider is the chart margin. When using area series, the XValues extend to the very edges of the chart space, so that the areas fill the entire chart space. But when using column, bar, or line series, the XValues are offset from the edges of the chart space, so that the first and last XValues in the chart do not bump up against the chart edges. When you combine an area series with one of the other series kinds, the margins for all series will be offset, which looks odd with area series.

Series Kind	Design Considerations for Combination Charts
Bar	• The first series in the data source must be a bar series in order to maintain the flipped orientation of the chart. If the first series is line or area, then the orientation will be based on that series, and the bar series will instead be displayed as a column series. • It is not possible to combine column and bar series, as they are basically the same type of series with different orientation. If a chart has column and bar series, the chart will interpret all of these series as the first type it encounters. For example, if the first series in the data source is a column, then all subsequent column or bar series will be columns.
Bubble	No special design considerations to note.
Column	It is not possible to combine column and bar series, as they are basically the same type of series with different orientation. If a chart has column and bar series, the chart will interpret all of these series as the first type it encounters. For example, if the first series in the data source is a column, then all subsequent column or bar series will be columns.
Line	No special design considerations to note.
Scatter	No special design considerations to note.
ScatterLine	No special design considerations to note.
Waterfall	Waterfall charts are very different than other XYChart types. In some cases you might want to display a line series with a waterfall series, to indicate an average or trend for comparison, but other XYChart types do not combine well with waterfall series.

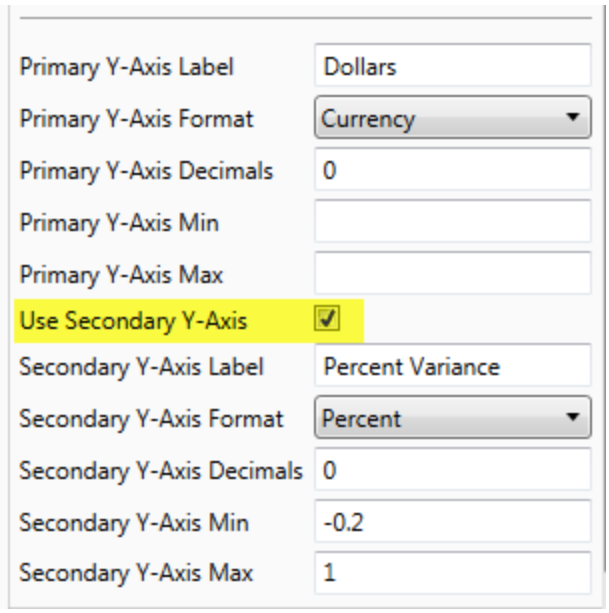
Using two Y-axis scales with combination XYCharts

By default, XYCharts have one primary Y-axis scale along the left-hand side of the chart. If desired, you can use a second Y-axis scale along the right-hand side of the chart, and then specify which scale each series uses. This is primarily used with combination charts, so that one series type can use the primary Y-axis, and the second series type can use the secondary Y-axis.

To use a secondary Y-axis, you must:

- Enable a secondary Y-axis for the chart by selecting the **Use Secondary Y-Axis** check box. This exposes a second set of Y-axis properties for the secondary axis. You can then define the properties for both the primary and the secondary Y-axis as appropriate.
- In the data source for the chart, add an `[Axis]` tag to the control row. For each series, enter either `Primary` or `Secondary`, depending on which Y-axis the series should belong to. If a series does not have a tag in the Axis column, then the primary axis is assumed.

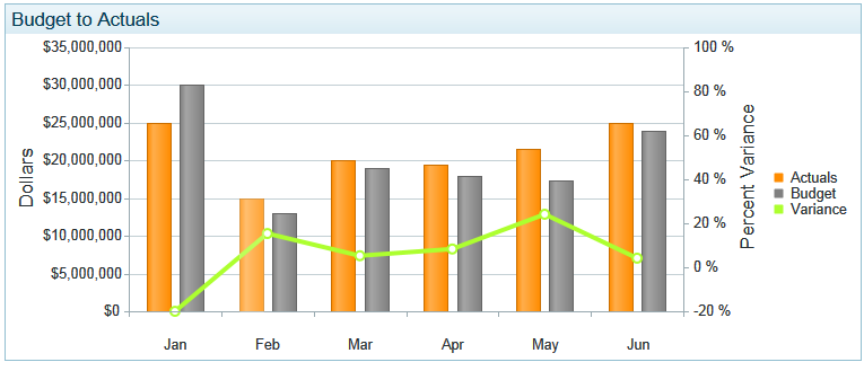
For example, you may have a chart with two column series that show dollars, and then you want a third line series that displays a variance percent. The properties for the chart would appear as follows:



The data source for the chart would appear as follows:

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		[XYChart;BudgetToActuals]	[Kind]	[SeriesName]	[Axis]	[Color]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]					Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Column	Actuals	Primary	#FFFF8C00	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Column	Budget	Primary	#FF808080	30000000	13000000	19000000	18000000	17400000	24000000
6							-5000000	2000000	1000000	1500000	4200000	1000000
7		[Series]	Line	Variance	Secondary	#FFADFF2F	-0.17	0.15	0.05	0.08	0.24	0.04
8												

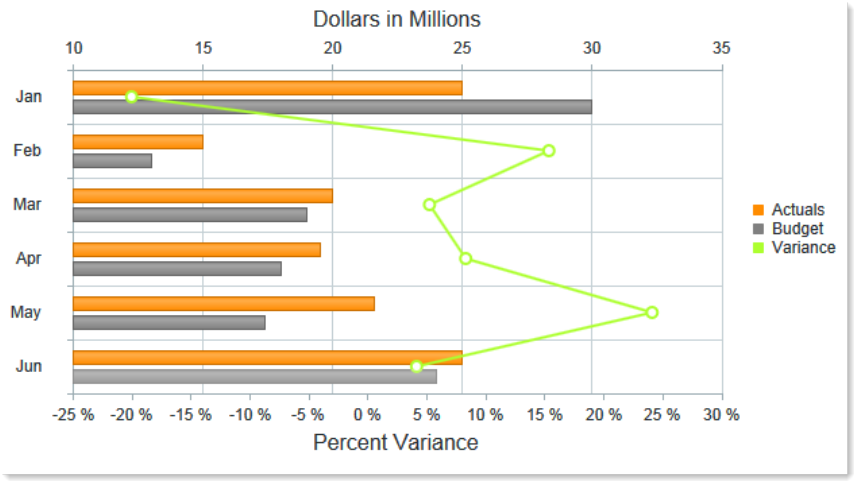
The resulting chart would appear as follows:



Note that if one of the series in the combination chart is a bar chart, then the chart is flipped so that the two Y-axes display along the top and bottom instead of the left and right. For example, the following screenshot shows a combination data source with bar and line series.

	A	B	C	D	E	F	G	H
1								
2		[XYChart;Axis]	[Kind]	[SeriesName]	[Axis]	[Color]	[XValue]	[XValue]
3		[XValueName]					Jan	Feb
4		[Series]	Bar	Actuals	Primary	#FF8C00	25	15
5		[Series]	Bar	Budget	Primary	#808080	30	13
6							-5	2
7		[Series]	Line	Variance	Secondary	#ADFF2F	-20%	15%

The resulting chart would appear as follows:



Specifying chart colors

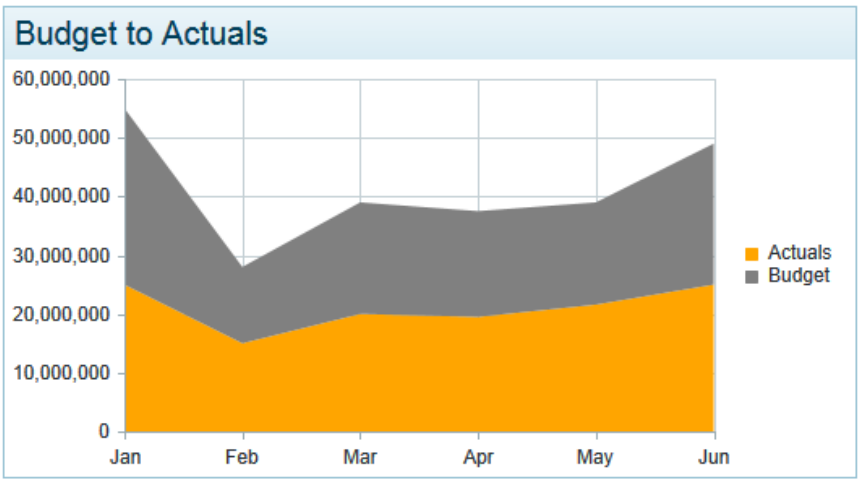
By default, when an Axiom form is viewed, colors are automatically assigned to each series in a chart based on a set of colors defined in the style or skin (in that order). Alternatively, you can assign a color to each series in a chart, and the chart will render using those colors. This applies to the following charts: all charts in the XYChart family, all charts in the ScatterChart family, and Pie Charts. Other charts have different methods of specifying colors.

In the data source for the chart, add a `[Color]` tag to the control row, and then enter the desired color for each series. You can use basic color names (for example, Blue) or you can enter valid hexadecimal color codes (for example #00FFFF for Aqua).

The data source for the chart would appear as follows:

	A	B	C	D	E	F	G	H	I	J	K
1											
2		[XYChart;Colors]	[Kind]	[SeriesName]	[Color]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]	[XValue]
3		[XValueName]				Jan	Feb	Mar	Apr	May	Jun
4		[Series]	Area	Actuals	Orange	25000000	15000000	20000000	19500000	21600000	25000000
5		[Series]	Area	Budget	Gray	30000000	13000000	19000000	18000000	17400000	24000000

The resulting chart would appear as follows:



For Pie Chart components, the color is defined per `[PieItem]` instead of per `[Series]`, but otherwise the color assignment works the same way.

Note the following design considerations for certain chart types:

- **ScatterChart**: Charts in the ScatterChart family can have multiple rows per series. In this case, the color assignment is handled as follows:
  - To use the same color for all items in the series, you can enter the same color on all rows of the series, or you can enter a color on only one row of the series (leaving all other rows in the series blank).
  - To use different colors on all items in the series, enter different colors on each row. This only applies to Bubble series and Scatter series. ScatterLine series can only use one color per line. If different colors are specified for the same ScatterLine series, the first color listed will be used.
- **Waterfall**: Waterfall charts support additional columns of `[RunningTotalColor]` and `[TotalColor]`, so that you can optionally specify different colors for columns showing computed totals.

Using Shapes

The shape components can be used to draw basic shapes on an Axiom form. In the Form Designer, these components are available in the **Shapes** section along the left-hand side of the screen.

- Ellipse: Draw a circle or ellipse.
- Horizontal Elbow Line: Draw a horizontal line or arrow with elbow bends at each end.
- Rectangle: Draw a square or rectangle.
- Straight Line: Draw a straight line or arrow.
- Vertical Elbow Line: Draw a vertical line or arrow with elbow bends at each end.

You can configure the fill and border color for shapes, as well as properties such as dashed lines and rounded corners. Shapes can also be used to launch a designated URL when clicked.

NOTE: Your Axiom Software license determines whether you have access to shape components. For more information, see Licensing requirements for Axiom forms.

Ellipse component

Use the Ellipse shape component to draw a circle or ellipse on the Axiom form. The shape can have color and text, and can be linked to a URL.

NOTE: Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.

Ellipse properties

You can define the following properties for an Ellipse component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the component. Leave this blank if you want to use a shape with no text.
Fill Color	The fill color for the shape. If left blank, the background color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color. If the component is inheriting a fill color from the style or skin and you want the component to use no background color, then you can specify `transparent` as the color. You must manually edit the Form Control Sheet to do this, because the Fill Color selector in the Form Designer / Form Assistant does not allow this entry.
Dashed Border	Specifies whether the border is dashed or solid. By default this is not selected, which means the border is solid. If you want a dashed border, select this check box.
URL	Optional. The URL to launch when a user clicks on the component. The URL must use full HTTP syntax—meaning, use `HTTP://www.axiomepm.com`, not `www.axiomepm.com`.
Use New Window	If a URL is defined, specifies whether the link is opened in a new window. By default this is enabled, which means the link is opened in a new window. Disable this option if you want the link to open within the same window (replacing the current Axiom form).

All shape components use the same basic properties. When viewing ellipse properties on the Form Control Sheet, you will see several settings that do not apply to the component:

- **Default Shape Type**: This is set to Ellipse when the component is placed on the canvas. Generally speaking, Axiom Software does not support dynamically changing the shape type.
- **Line is Top Left**: This property only applies to Straight Line and Elbow Line components.
- **Rectangle Has Rounded Corners**: This property only applies to Rectangle components.
- **Line Kind / Line Arrow Kind**: These properties only apply to Straight Line and Elbow Line components.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Elbow Line component

Use the Elbow Line component to draw a bent line or arrow. The bends can extend from a horizontal line or a vertical line.

Although all three line types display as separate components in the Form Designer, all three use the same base component and have the same settings. The Line Kind determines the type of line, and is set automatically based on which component type you place on the canvas.

NOTES:

- Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.
- Lines are not compatible with **Scale to Fit**. If Scale to Fit is enabled, lines may not scale accurately.

▶ Line properties

You can define the following properties for a Horizontal Elbow Line or Vertical Elbow Line component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Line Color	The line color. If left blank, the color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.
Line Thickness	The thickness of the line, in pixels. If left blank, the thickness is determined by the style or skin (in that order). The specified or inherited thickness must be greater than 0 or else the line will not display on the form.
Dashed Line	Specifies whether the line is dashed or solid. By default this is not selected, which means the line is solid. If you want a dashed line, select this check box.
Arrow Kind	Specifies the arrow options for the line. Select from the following: • **None** (default): No arrow head is present on the line. • **Right**: Place an arrow head on the right end of the line. • **Left**: Place an arrow head on the left end of the line. • **Both**: Place an arrow head on both ends of the line.
Line Kind	Specifies the type of line. By default, this is determined based on which component type you place on the canvas; however, you can change the type later if desired: • **Straight** • **Horizontal Elbow** • **Vertical Elbow**
Line is Top Left	Determines where the line starts. By default this is not selected, which means the line starts at the bottom left. Select this check box if you want the line to start at the top left.

All shape components use the same basic properties. When viewing line properties on the Form Control Sheet, you will see several settings that do not apply to the component:

- **Default Shape Type**: This is set to Line when the component is placed on the canvas. Generally speaking, Axiom Software does not support dynamically changing the shape type.
- **Fill Color**: This property only applies to Ellipse and Rectangle components.

- **Rectangle Has Rounded Corners**: This property only applies to Rectangle components.
- **URL / Use New Window**: These properties only apply to Ellipse and Rectangle components.
- **Text / Text Color / Font Family / Font Size / Font Weight / Font Style**: These properties only apply to Ellipse and Rectangle components.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Rectangle component

Use the Rectangle shape component to draw a square or rectangle on the Axiom form. The shape can have color and text, and can be linked to a URL.

NOTES:

- Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.
- If you want to display a background color on the entire form, then you should use the **Background Color** property at the form level instead of placing a Label or Rectangle component on the canvas and sizing it to fit the entire area. For more information, see Setting the background color or image for an Axiom form.

▶ Rectangle properties

You can define the following properties for a Rectangle component.

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Text	The display text for the component. Leave this blank if you want to use a shape with no text.
Fill Color	The fill color for the shape. If left blank, the background color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color. If the component is inheriting a fill color from the style or skin and you want the component to use no background color, then you can specify `transparent` as the color. You must manually edit the Form Control Sheet to do this, because the Fill Color selector in the Form Designer / Form Assistant does not allow this entry.
Dashed Border	Specifies whether the border is dashed or solid. By default this is not selected, which means the border is solid. If you want a dashed border, select this check box.
Rounded Corners	Specifies whether the corners of the component are rounded. By default this is not selected, which means regular (90 degree) pointed corners. If you want rounded corners, select this check box.
URL	Optional. The URL to launch when a user clicks on the component. The URL must use full HTTP syntax—meaning, use `HTTP://www.axiomepm.com`, not `www.axiomepm.com`.
Use New Window	If a URL is defined, specifies whether the link is opened in a new window. By default this is enabled, which means the link is opened in a new window. Disable this option if you want the link to open within the same window (replacing the current Axiom form).

All shape components use the same basic properties. When viewing rectangle properties on the Form Control Sheet, you will see several settings that do not apply to the component:

- **Default Shape Type**: This is set to Rectangle when the component is placed on the canvas. Generally speaking, Axiom Software does not support dynamically changing the shape type.
- **Line is Top Left**: This property only applies to Straight Line and Elbow Line components.

- **Line Kind / Line Arrow Kind**: These properties only apply to Straight Line and Elbow Line components.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Straight Line component

Use the Straight Line component to draw a straight line or arrow on the Axiom form canvas.

All line components use the same settings. The Line Kind determines the type of line, and is set automatically based on which component type you place on the canvas.

NOTES:

- Your Axiom Software license determines whether you have access to this component. For more information, see Licensing requirements for Axiom forms.
- Lines are not compatible with **Scale to Fit**. If Scale to Fit is enabled, lines may not scale accurately.

Component properties

You can define the following properties for a Straight Line component:

Component behavior properties

The following properties control the display and behavior of this particular component type.

Item	Description
Line Color	The line color. If left blank, the color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color.
Line Thickness	The thickness of the line, in pixels. If left blank, the thickness is determined by the style or skin (in that order). The specified or inherited thickness must be greater than 0 or else the line will not display on the form.
Dashed Line	Specifies whether the line is dashed or solid. By default this is not selected, which means the line is solid. If you want a dashed line, select this check box.
Arrow Kind	Specifies the arrow options for the line. Select from the following: • **None** (default): No arrow head is present on the line. • **Right**: Place an arrow head on the right end of the line. • **Left**: Place an arrow head on the left end of the line. • **Both**: Place an arrow head on both ends of the line.
Line Kind	Specifies the type of line. By default, this is determined based on which component type you place on the canvas; however, you can change the type later if desired: • **Straight** • **Horizontal Elbow** • **Vertical Elbow**
Line is Top Left	Determines where the line starts. By default this is not selected, which means the line starts at the bottom left. Select this check box if you want the line to start at the top left.

All shape components use the same basic properties. When viewing line properties on the Form Control Sheet, you will see several settings that do not apply to the component:

- **Default Shape Type**: This is set to Line when the component is placed on the canvas. Generally speaking, Axiom Software does not support dynamically changing the shape type.
- **Fill Color**: This property only applies to Ellipse and Rectangle components.

- **Rectangle Has Rounded Corners**: This property only applies to Rectangle components.
- **URL / Use New Window**: These properties only apply to Ellipse and Rectangle components.
- **Text / Text Color / Font Family / Font Size / Font Weight / Font Style**: These properties only apply to Ellipse and Rectangle components.

General properties

All form components support a set of general properties such as component name and layer. For more information on these properties, see General properties.

Style and formatting properties

All form components support the ability to assign styles to determine the component formatting, such as fonts, borders, and colors. For some components, you can also define certain formatting properties at the component level, overriding any properties set by the skin, theme, or styles. For more information on these properties, see Style and formatting properties.

Position and size properties

All form components use a set of standard position and size properties. These properties control where each component displays on the form web page and how much space each component takes up. For more information on these properties, see Position and size properties.

Setting the slope of a line

Once you place a line on the canvas, you can drag a selection handle to make the overall component bigger or smaller, but you cannot change the slope of the line by dragging. To change the slope of a line, you must manually adjust the width or height. To access these properties, click **Show Advanced Settings** under the **Style** box.

For example:

- If you want the line to be horizontal, set the **Height** to 0px.
- If you want the line to be vertical, set the **Width** to 0px.

You can further adjust the slope by adjusting the width or height as desired. For example, if 0px height is fully horizontal, then 5px height is a gentle slope upward, and 20px is a slightly steeper slope.

The setting **Line is Top Left** determines the direction of the slope. If disabled, the line slopes upward from left to right. If enabled, the line slopes downward from left to right.

Using Axiom Forms for Planning

This section discusses Axiom form features that are only available if the source file for the Axiom form is a template / plan file. In most cases, the Axiom form is configured at the template level, and then all plan files generated from that template inherit the form setup.

Considerations when using Axiom forms as plan files

You can use form-enabled Axiom files as plan files in a file group. This topic discusses some considerations when using this configuration.

Opening form-enabled plan files

Form-enabled plan files can be accessed using either the Web Client or the Desktop Client (Excel Client and Windows Client).

If end users will access plan files using the Web Client, you can provide this access using one of the following approaches:

- The recommended approach is to configure the users' form-enabled home page to include a hyperlink to the relevant Plan File Directory page. (This hyperlink could also be placed on a similar form that is used as a launching point for the particular planning activity.) The user can click the link to be taken to a system-generated list of plan files that is filtered to only show the plan files that the current user has access to. The user can open their plan files from this page. This approach provides a user interface that is similar to the Open Plan Files dialog in the Desktop Client.

  For more information on how to link to this page and how to configure this page, see Using the Plan File Directory page.

- Alternatively, you can manually generate a list of hyperlinks to plan files and include it on the users' form-enabled home page (or similar form). For example, you could use an Axiom query to populate a Formatted Grid component with a list of plan codes (such as departments), and then use the HREF content tag to create hyperlinks to the corresponding plan files. You would need to be able to filter the query so that it only shows the plan files that the user has access to. This approach takes more work to set up, but allows you to present the list of plan files however you like.

- Lastly, users can use the built-in browse page for Axiom forms in the Web Client. Keep in mind, however, that this page lists all forms that the user has permission to access, including reports. Forms are organized by folder, so the user would have to navigate to the appropriate file group folder in order to access their plan files.

If end users will access plan files using the Desktop Client, they can access these form-enabled plan files using the normal Open Plan Files dialog for the file group. If a plan file is form-enabled, it will automatically open as a web form in the user's browser (when using the Excel Client), or as a web tab within the application (when using the Windows Client).

If you need to open a form-enabled plan file as a spreadsheet in the Desktop Client, you can right-click the plan file in the Open Plan Files dialog and choose **Open as Spreadsheet**. This should be a rare occurrence since any modifications to the source spreadsheet file should be made in the template, not in the individual plan files. However you may need to occasionally open a form-enabled plan file as a spreadsheet for troubleshooting.

Security configuration for form-enabled plan files

When configuring security for form-enabled plan files, keep in mind the following:

- Axiom forms are always opened read-only, so read/write access is irrelevant for end users. However, read/write access is still required to allow upload of plan file attachments, if applicable.
- **Allow Save Data** must still be enabled if you want the user to save data. Note that process ownership can be used to elevate users to this permission as normal.
- **Allow Calc Method Insert** must still be enabled if you want users to be able to insert calc methods, assuming that the Axiom form is configured to support inserting calc methods.
- All other special permissions are irrelevant in the Axiom form context (such as calc method change, unprotect, etc.).

If you are using composite forms as plan files, then end users must also be granted access to the child forms that are included in the parent plan file. For more information, see Designing composite plan files with Axiom forms.

Creating plan files

For standard file groups, form-enabled plan files are created as normal, using the Create Plan Files utility. There are no special considerations in this case.

Special considerations may apply to on-demand file groups, depending on the method of plan file access.

- If plan files will be created using the Desktop Client (Excel Client or Windows Client), then all normal on-demand features can be used to create plan files. The only time special considerations apply is when you must collect starting values for the plan code table, for columns that are *not* lookup columns or the template column. The Desktop Client will automatically prompt the user to select values for lookup columns and the template column. If you need to collect values for other columns, then you will need to create an Axiom form to collect these values as part of the plan file

creation process, and assign this form as the **Add File Form** in the file group properties.

- If plan files will be created using the Web Client, then you must provide users with the means to create plan files by configuring an Axiom form with the Add Plan File command. Additionally, if starting values are necessary for certain columns in the plan code table, you must configure the Axiom form to collect these values and pass them to the table using the Add Plan File command—the Web Client does not provide any means to automatically collect these values for any column.

  You can provide a link to this "plan file creation" form on the users' form-enabled home page (or similar form). You can also designate this form as the **Add File Form** for the file group, which means that users can access the form from the Process Summary component or from the Plan File Directory page.

Access to plan file features

When using form-enabled plan files, keep in mind that users will not have access to most Axiom Software features that are available when using spreadsheet plan files. Users will also not have access to spreadsheet features such as cell formatting, creating formulas, etc.

Axiom form users only have access to planning features that are supported for use in Axiom forms, and even then only if the form is configured to use them. For example, Axiom forms support inserting new rows using calc methods, but if the form itself is not configured with this functionality, then it will not be available to the user. There are no "built-in" planning features for Axiom forms; users can only do what the form is configured to allow them to do.

In some cases, you can manually create form equivalents to certain features that are available in spreadsheet Axiom forms. For example, although the Change View feature is not available in Axiom forms, you can simulate this feature by configuring certain row and column tags in a Formatted Grid to dynamically show and hide. You can then set up an interactive component, such as a combo box or radio buttons, to allow form users to switch between these "views".

Designing composite plan files with Axiom forms

When using form-enabled plan files, it is possible to design *composite plan files*. A composite plan file means that the contents of the plan file are sourced from multiple separate files instead of all within a single template. These separate files are displayed as embedded within the plan file form, creating a parent/child relationship between the files.

The benefits of this approach include:

- Simplifies form design by breaking up contents into discrete units.
- Enables dynamic content for the plan file.
- Enables re-use of the child forms (for example, in multiple related plan file templates).

Instead of needing to include every possible variation of content within a single file (the plan file template), all of the needed content can be defined in separate utility files. To dynamically source the

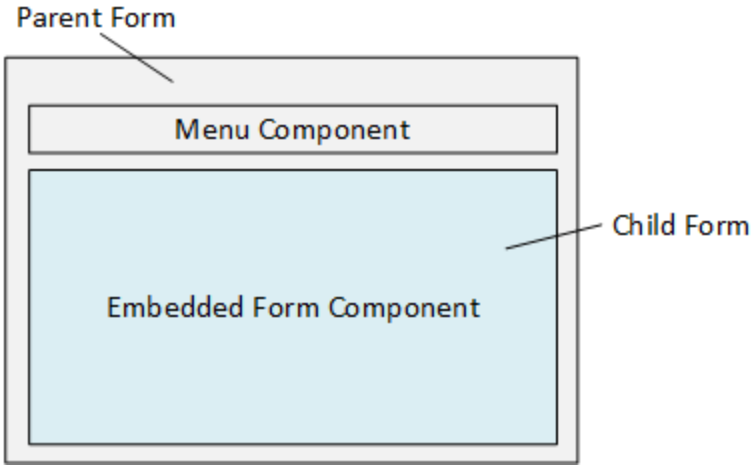
content, the template can be configured so that it opens a particular content file or not, depending on some factor particular to each plan file.

The composite form design can be used for any form, not just plan files. For more information on creating composite forms—including how to share information between forms and how to manage the save and update process—see Using composite forms. This topic discusses the design considerations unique to creating composite plan files.

Setting up composite plan files

In order to use composite plan files, the plan file template and the child files must be set up as follows:

- The form-enabled template must contain a Menu component to allow switching between multiple child files, and an Embedded Form component to display the currently selected child file. The parent template itself usually does not have much content, though it may display overall titles and brief summary content in addition to the menu and embedded form. For more information on using these components, see Menu component and Embedded Form component.



- The child files must be form-enabled utility files within the same file group as the template. This is necessary so that all child files have the same file group context as the template / plan files, and so that the child files and templates will be kept together as the file group is cloned or for scenario creation.
- The template and the child files must be set up with shared variables as appropriate, to share values between all the files in the shared form instance. At minimum, the template must define a shared variable for the current plan code, and the child forms must reference that variable value, so that all files are retrieving and saving data for the current plan code. For more information, see Sharing variables between parent and child forms.

In most cases, **Use Virtual Plan Files** should be enabled for the file group. This means that physical plan files are not stored in the system, and instead the plan files are dynamically created from the template each time they are accessed. Although the plan files could be persistent if desired, there is not much to be gained from storing the physical files because files are not saved in the forms environment (only data),

and because most of the plan file contents are sourced from the child files instead of the plan file template itself.

When designing the contents of the parent and child forms, make sure that you understand the update behavior for plan files in a shared form instance, especially in terms of understanding when save-to-database is triggered. For more information, see Form session and update behavior for composite forms and Saving data from composite forms.

End user experience

When a user opens a composite plan file, the parent plan file opens and displays the child form that is designated as the starting selected ID for the Menu component. The user can interact with the currently displayed child form, or the user can switch to another child form by using the menu.

As the user switches from child form to child form, the state of each child form is maintained on the Axiom Application Server, even if that form is not currently visible. So it is not required to save data before switching child forms, unless you want to do so for design purposes (for example, if one child form queries data that is saved from another child form).

Security considerations for composite plan files

When using composite plan files, users with access to the plan files also need access to the child forms (the utility files). Plan file permission sets defined on the **File Groups** tab in Security are not sufficient to view the child form contents, because they only grant permission to the plan file itself. Process management and workflow ownership will not grant or "elevate" access to the child forms.

On the **Files** tab of Security, users with access to the plan files should also be granted the following access to the **Utility** folder of the relevant file group (either at the user or role level):

- **Read-Only** access
- **Allow Save Data** is enabled
- **Show in Explorer** is disabled

This configuration allows the user to open the utility files using the references within the plan file, but the utility files will not display in file explorer views throughout the system, such as in the Explorer task pane in the Desktop Client, or in the Forms browse page in the Web Client. This is most likely the desired level of access for these users. If a user needs a different level of access, it can be configured as needed.

Processing composite plan files

If you want to use Process Plan Files with composite plan files, the plan file template must be set up to enable "processing with utilities." Because the planning content for composite plan files is contained in utility files instead of within the plan files, using traditional plan file processing would not update planning data. Instead, you want to be able to process the child utilities using each plan code.

To accomplish this, you create a ProcessPlanFileUtilities data source in the template. This data source lists the child utilities to be processed and the processing order.

When using Process Plan Files with composite plan files, select the **Process with Utilities** processing mode. For each plan file to be processed, the plan file is opened, refreshed, and shared variables are set. The list of utilities to process is read from the data source in the plan file. For each enabled utility in the data source, the shared variables are passed into the utility, then the utility is refreshed and a save-to-database occurs. The plan code is passed from the plan file to the utilities using the shared variables (along with any other necessary information), so that the utilities can be filtered as needed for the current plan code. Using the shared variables for this purpose means that you can leverage the same variable setup for processing that you do when users work within the live form.

To make "processing with utilities" the default processing option for a file group, enable **Process Plan Files with Utilities** in the file group properties. This means that Process Plan Files will default to utility processing, and plan file process definitions will use utility processing to validate plan files if **Save and validate plan file before advancing to next step** is enabled for a process step.

For more information on utility processing and using process plan files, see the *File Group Administration Guide*.

Creating new on-demand plan files using an Axiom form

You can set up an Axiom form to allow users to create new plan files for an on-demand file group. You might do this so that users can create new plan files using the Web Client environment, or you may want to use the Axiom form as an alternative "dialog" for plan file creation within the Desktop Client.

Requirements and limitations

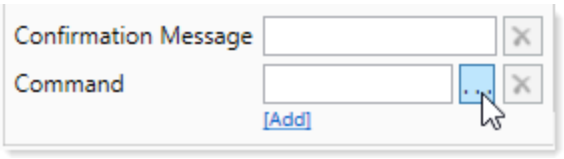
The Add Plan File command for use in Axiom forms supports creating new on-demand plan files based on a template (either the default template for the file group, or an entry in the designated template column). It is also possible to use the Add Plan File command to clone an existing plan file, but only when the form is used as a Clone File Form within the Excel Client or Windows Client (see Using an Axiom form as an "add file" dialog in the Desktop Client below).

Setting up new plan file creation in an Axiom form

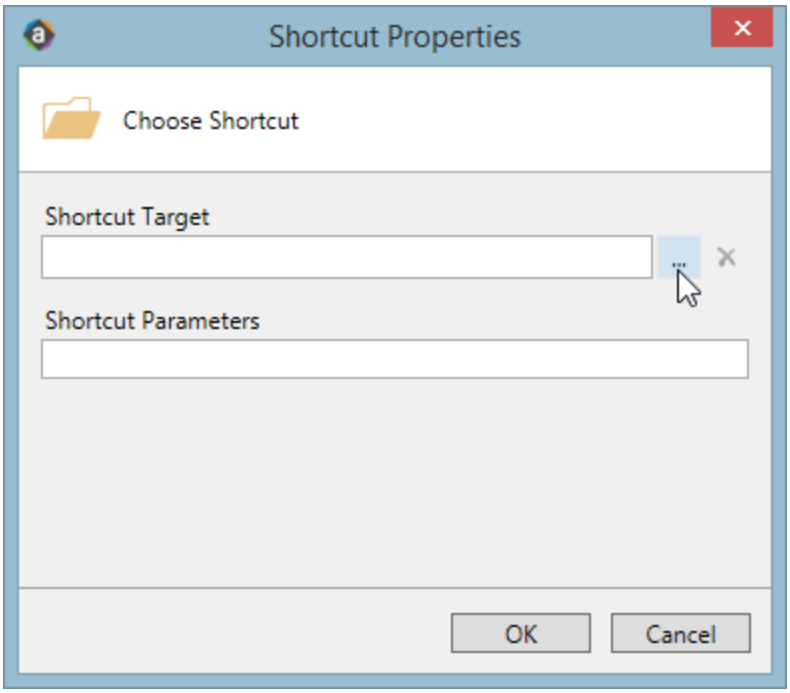
To allow users to create new plan files from an Axiom form, you use a Button component that is configured to run the Add Plan File command.

To start off, add a Button component to the Axiom form canvas and then configure the button properties as desired. You will probably want the display text to be something like "Create New Capital Request" or "Create New Strategic Plan" (or whatever plan type the file group is for). You can then configure the **Command** for the component as follows:

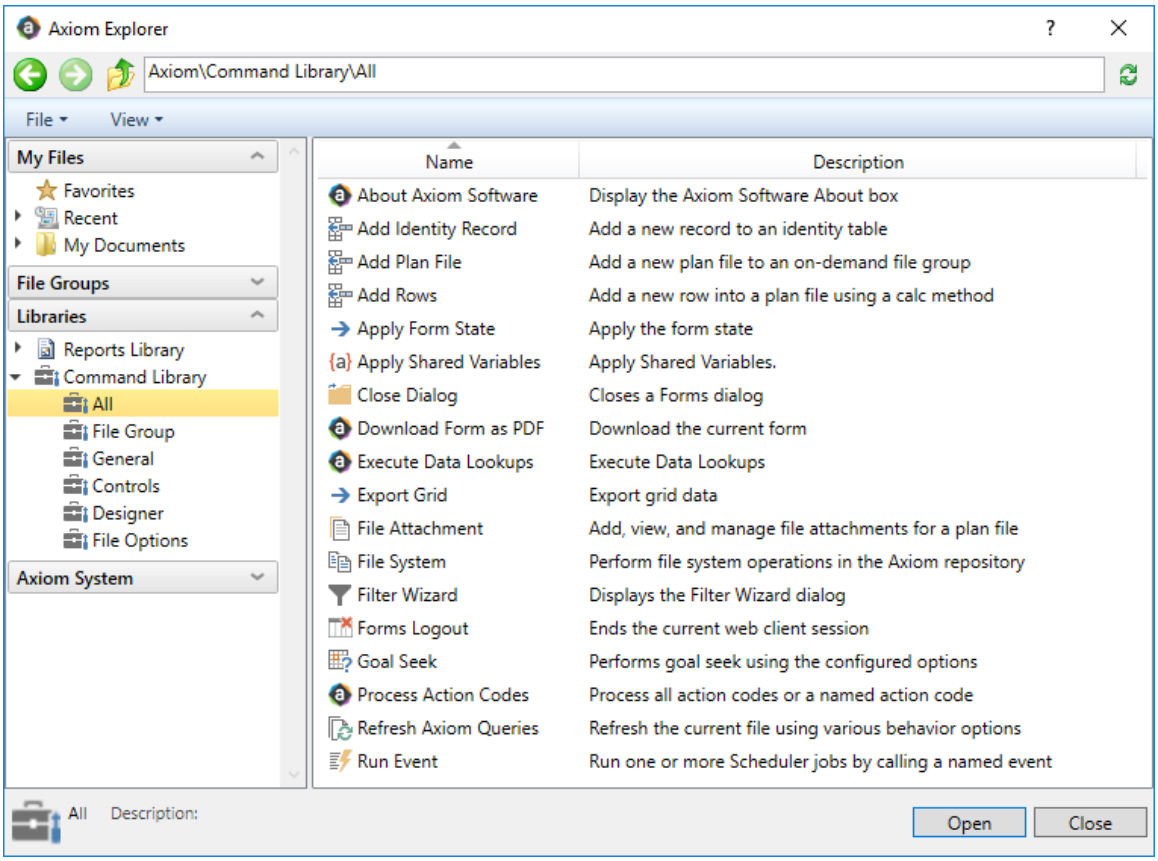
1. In the component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



3. In the **Axiom Explorer** dialog, select **Command Library > All**, then select the **Add Plan File** command. Click **Open**.



In the Shortcut Properties dialog, the Add Plan File command is now listed as the shortcut target, and the relevant shortcut parameters are now available.

4. In the **Shortcut Parameters**, complete the following settings:

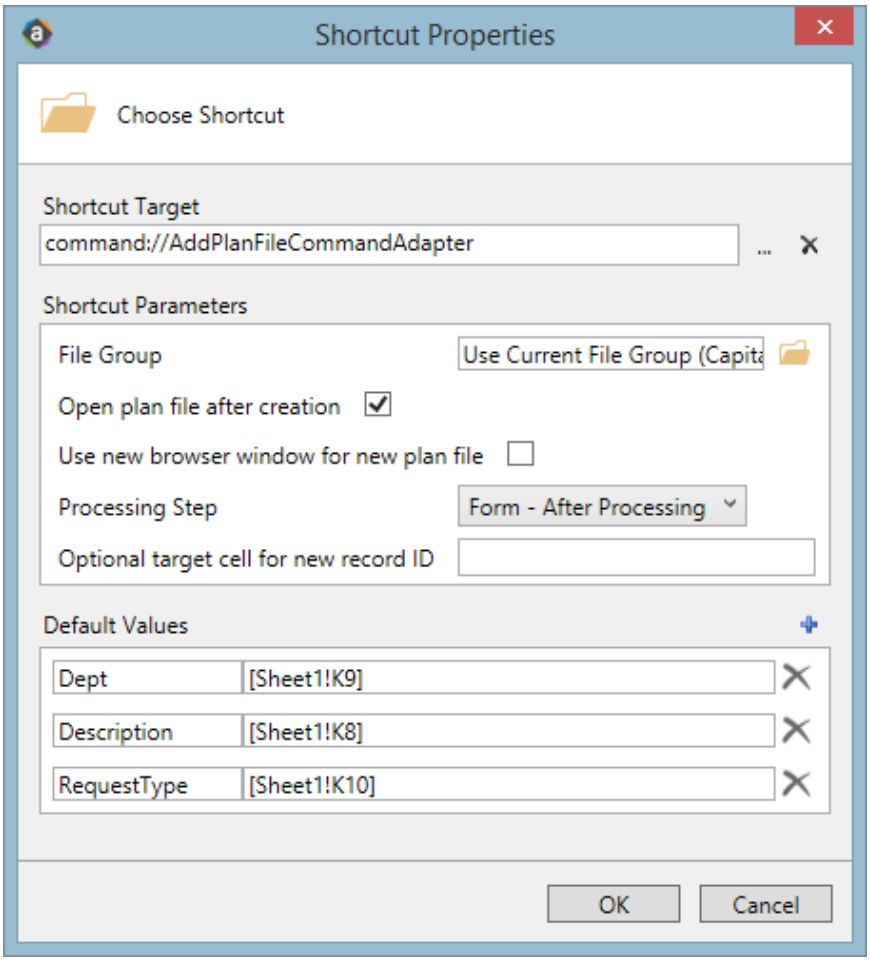
Item	Description
File Group	Select the on-demand file group for which you want users to be able to create new plan files. A file group alias can also be used. **NOTE:** If the source file for the Axiom form is a file group utility, then it is strongly recommended to use the **Use Current File Group** option instead of selecting a specific file group. The button will dynamically use the file group that the utility file belongs to. This is especially useful when cloning the file group, so that the cloned utility file automatically points to the new file group.

Item	Description
Open plan file after creation	This option is selected by default, which means that when a new plan file is created, it will automatically be opened. You might clear this check box if end users do not need to work with the new plan files they have created. For example, end users might use the Axiom form to simply submit an initial request (the new plan file), which is then reviewed by a different user before any further work is done.
Use new browser window for new plan file	Select this option if you want the new plan file to open in a new browser window (or tab, depending on the browser settings) instead of opening in the current window. This option only applies when the new plan file is form-enabled, and when the command is being used from within a browser (as opposed to a form dialog within the Desktop Client).
Processing step	Optional. Specify the desired **Processing Step** for the command. By default, this is set to **Form - After Processing**, which means that the command will be performed after the entire form update process is complete. If desired, you can specify a different processing step for the command. For more information, see Timing of command execution and Axiom form update process. **NOTE:** If the command is configured to execute at an earlier processing step, the actions of adding the new record and creating the plan file will occur at the specified processing step. However, if the command is configured to open the new plan file after creation, that action will continue to occur at the After Processing step.
Optional target cell for new record ID	Optional. Specify a target cell in the source file to place the ID of the new record after it has been created. For example: `Info!A1`. If a target cell is specified, then the source file will be calculated after the value is placed in the cell. You might use this if you want to first create the new record / plan file, then reference that new record number in a save-to-database process to a different table.

5. If you need to populate columns in the plan code table when the new record for the plan file is created, you can do this using the **Default values** section.

- Enter the column name in the left-hand box. This represents the column in the plan code table that you want to populate when creating a new record. Do *not* use the fully-qualified column name—for example, enter `Dept`, *not* `CapitalID.Dept`.
- Enter the value for the column in the right-hand box. You can enter a "hard-coded" value, or you can enter a cell reference in brackets to read the value from that cell. See the following section Collecting starting values from an Axiom form for more information on how to use the cell references with form components.
- If you need more rows to define additional default values, click the plus icon ✚.

These values will be saved to the plan code table when the new record is created. At minimum, you must include all alternate key columns (if any) and the designated Template column (if applicable to the file group). You can include any other columns (including validated columns) as desired. If you do not include a validated column, then that column must have a valid default value defined in its column properties, and that value will be used when the record is created.



Example Shortcut Properties dialog

6. Once you have finished configuring the Shortcut Properties, click **OK** to close the dialog and return

to the component properties.

When the Axiom forms user clicks the Button component, Axiom Software first creates the new record in the plan code table, and then creates the new plan file. After the plan file is created, the behavior depends on the **Open plan file after creation** setting:

- If **Open plan file after creation** is disabled, then a message displays to the user to inform them that the plan file was created.
- If **Open plan file after creation** is enabled, then the new plan file is opened. Note the following:
  - If the new plan file is not form-enabled, then it will be opened as a spreadsheet in the Desktop Client. If the current form is open in the Web Client, and the Desktop Client is not installed on the current machine (or if the form is open on a device that does not support the Desktop Client), then the new plan file cannot be opened. If this situation is likely to occur in your environment, it is best to disable **Open plan file after creation**.
  - If the new plan file is form-enabled, then it will open according to the option **Use new browser window for new plan file**. If this option is disabled and the current form is open in the Web Client, the new plan file will open in the current window (replacing the current form contents). If this option is disabled and the current form is open as a dialog in the Desktop Client, then the new plan file will open as a web tab in the Desktop Client instead of in the browser. However, if you are using Excel 2013 or 2016, then the new plan file will always open in the browser.

Collecting starting values from an Axiom form

You can set up the Axiom form to collect starting values from the user, and then use those values when creating the new record in the plan code table for the new plan file. For more information on why you might want to do this, see the discussion on on-demand file groups in the *File Group Administration Guide*.

To do this, you must set up the Axiom form as follows:

- Place one or more interactive components in the Axiom form to collect the input from the user. For example, you might use a Text Box or Combo Box component, or you might use content tags within a Formatted Grid component.
- When configuring the Default Values section for the Add Plan File command, you should designate a cell reference in brackets as the value. Axiom Software will read the value from the designated cell.

If the component that you are using to collect the user input is a Formatted Grid component, then the cell reference in the Default Values section can simply point to the target cell of the content tag. For example, if the grid contains a Select tag where the target cell is A15 on sheet Values, then you enter `[Values!A15]` in the Default Values.

If you are using a different interactive component, such as a Text Box component, then you should use an indirect cell reference for the interactive property of the component, so that the value is written to

and read from a cell in another sheet. For example, for the Text property of the text box, you can enter `[Values!A15]`. This means that the text box value will be written to and read from that cell, instead of the Text cell on the Form Control Sheet. You would then also enter `[Values!A15]` in the Default Values. You should use this indirect behavior instead of referencing a cell on the Form Control Sheet directly, because any time a new component is added to or deleted from the form, that cell reference may change.

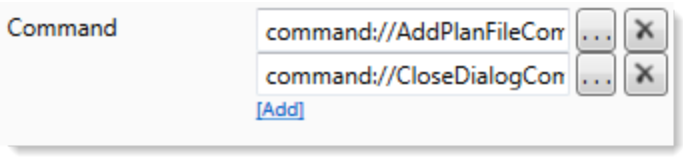
See the screenshot in the previous section for an example of the Default Values section with bracketed cell references.

Using an Axiom form as an "add file" dialog in the Desktop Client

If users are creating new plan files from within the Excel Client or the Windows Client, you can opt to use an Axiom form as the "input dialog" for collecting starting values for the plan code table (an Add File Form), instead of using the default dialog. For more information, see the discussion on on-demand file groups in the *File Group Administration Guide*.

If your form will be used as an Add File Form in the Desktop Client, the following additional design considerations apply:

- Set the **Canvas Size** of the form (width x height) to a dialog-appropriate size. A good starting point is 700 x 400. The maximum height is 800 for a form dialog; if the form is larger than that then a scroll bar will be present.
- If you want the dialog to automatically close after the new plan file is created and opened, then you should add a **Close Dialog** command to the same Button component that is configured with the Add Plan File command. For more information, see Using multiple commands on a button and Configuring close options for a form dialog.



  If the Close Dialog command is not present on the button, then the user must manually dismiss the dialog after creating the plan file.
- It is recommended to include a separate "Cancel" button that is configured with the **Close Dialog** command. This allows the user to close the dialog without creating a new plan file. The user can also click the **X** button at the top right-hand corner of the dialog to cancel, but providing a separate Cancel button provides a more typical user experience.
- It is recommended to use a file group utility as the source file for the form, instead of a report file. File group utilities are part of the file group, which means that the file can be copied when cloning the file group, and the target file group for the Add Plan File command can be dynamically updated (using the **Use Current File Group** option).

- If desired, you can define a form title (in the Form Properties) and this title will display as the dialog title. If no form title is defined, the title of the dialog will be **Add New Record**.
- If the form is being used as a Clone File Form, you can return the ID of the plan file being cloned and then use that ID in formulas for the purposes of setting default values for the new plan file. To return the plan file ID, use the reserved key of `SourceID` in a GetFormState function. For example:

  ```
  =GetFormState("SourceID")
  ```

  This would return 43 if the automatically generated ID value for the plan file being cloned is 43.

  The SourceID value is passed to the Clone File Form when a user clicks **Clone Selected Item** in the Open Plan Files dialog.

Using file attachment features in an Axiom form

File groups can be configured to allow users to attach supporting files to plan files. For example, the file group could be for capital planning, and you want users to be able to attach supporting documents to a particular capital request.

When using form-enabled plan files in the Web Client, you can provide users with access to file attachment functionality for the current plan file as follows:

- By default, users can use the File Attachments panel in the Web Client to upload attachments, open attachments, and manage attachments. This feature is automatically available if file attachments are enabled for the file group.
- You can optionally use the Upload Plan File Attachment button behavior within a plan file to allow users to upload new attachments only.

Additionally, the following features can be used in any form-enabled file:

- You can use the File Attachment command on a button to allow users to upload attachments, open attachments, and manage attachments for a specified plan file.
- You can optionally query the Axiom.PlanFileAttachments table to generate a list of attachments, and then generate hyperlinks to open those attachments.

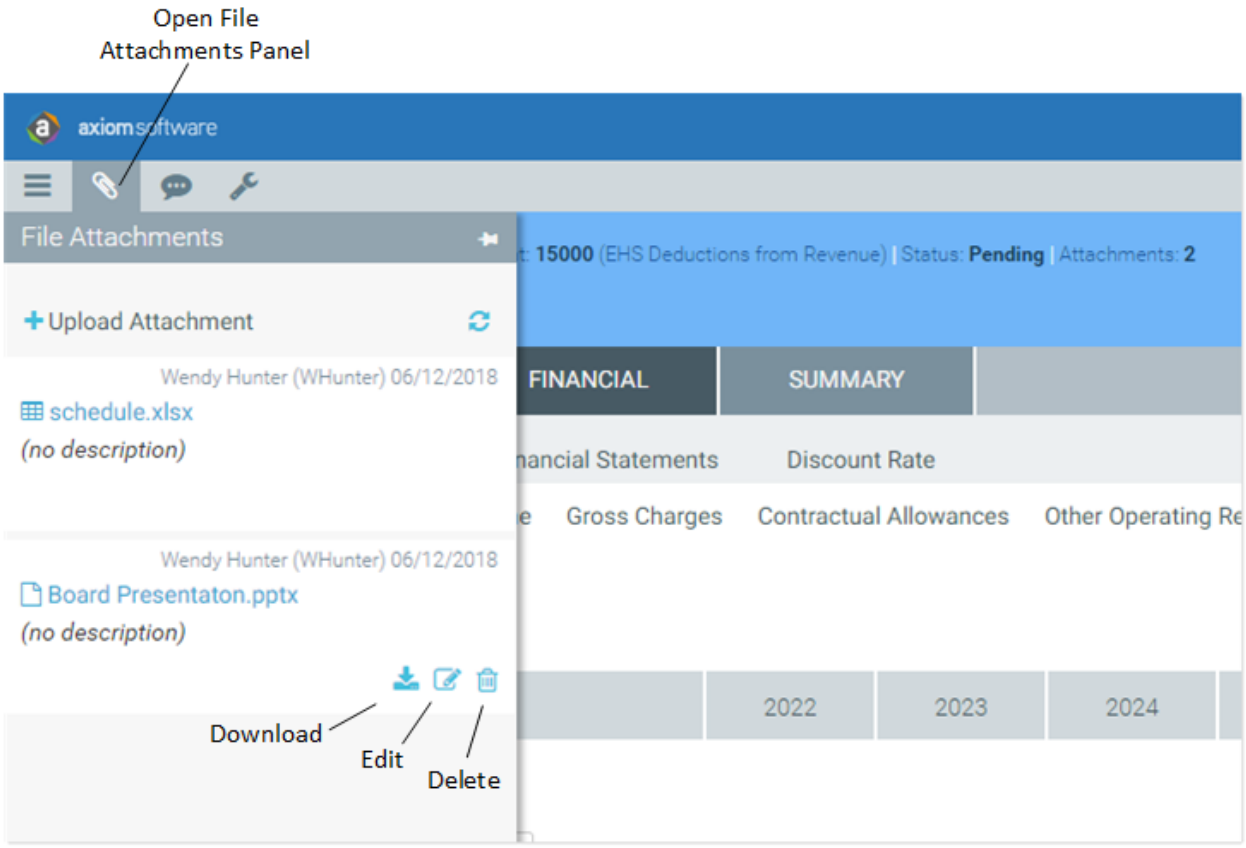
The user's security permissions to the plan file are applied as normal to determine whether users can upload and manage attachments, or just view existing attachments.

If a form plan file is open as a web tab in the Desktop Client, then users can use the regular **File Attachments** button on the Axiom ribbon tab to upload, open, and manage attachments. The behavior of the button is the same as for spreadsheet Axiom files. Forms can only be opened as web tabs within the Desktop Client when using the Windows Client.

Using the File Attachments panel to manage attachments

When a form-enabled plan file is open in the Web Client, users can use the File Attachments panel of the Web Client task bar to upload, open, and manage attachments for that plan file. The File Attachments panel is available automatically if attachments are enabled for the file group, and if the Using the Web Client Container with Axiom forms is enabled for the form plan file.

Users can click the paper clip icon in the top left of the task bar to open the File Attachments panel for the current plan file. If the plan file already has attachments, those attachments display in the panel.



Using this panel, users can perform the following actions:

- Upload a new attachment, or overwrite an existing attachment with a new version of the file
- View a list of all available attachments, as well as who uploaded them and when they were uploaded or modified
- Download and open a copy of the attachment
- Edit the attachment name or description
- Delete the attachment

All users with access to the plan file can view the list of existing attachments and download those attachments. In order to upload, edit, or delete an attachment, users must have read-write access to the

plan file. (Elevated access due to process step ownership is honored for purposes of determining the user's current access level to plan files.)

Using a button in a form to manage attachments

You can use the File Attachment command on a button to allow users to upload, view, and manage attachments associated with a specified plan file. Users can click the button to open the File Attachments dialog and perform file attachment actions according to their security permissions to the specified plan file.

This feature can be used in any form-enabled plan file. It can be used within the plan file itself as an alternative to using the built-in File Attachments panel in the Web Client Task Bar. It can also be used in a utility file or a report file to provide access to file attachments for one or more plan files.

For example, you might have a report that brings in information for all of a user's plan files in a particular file group, and you want to provide a way for that user to manage attachments for all of those plan files without needing to individually open each plan file. You can use a Button tag on each row of the Formatted Grid component, where the tag opens the File Attachments dialog for the current row's plan code.

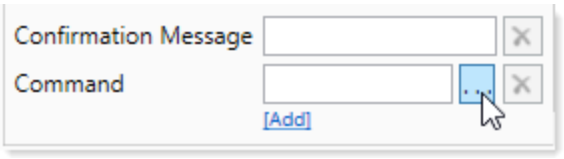
Requirements and limitations

- You can allow users to upload, view, and manage attachments for a specified plan file by using a button with the File Attachment command. The command can be used on a Button component or on a Button tag for a Formatted Grid component.
- The File Attachment command opens the Web Client File Attachments dialog for the specified plan file. This dialog displays attachments and attachment actions in a similar manner as the File Attachment panel in the Web Client Task Bar. The actions available in the dialog vary depending on the user's security permissions to the specified plan file.
- When the File Attachment command is used on a button, the normal form update cycle does not occur. The button only opens the File Attachments dialog. The command cannot be used with any other commands or any other update options (such as save on submit).

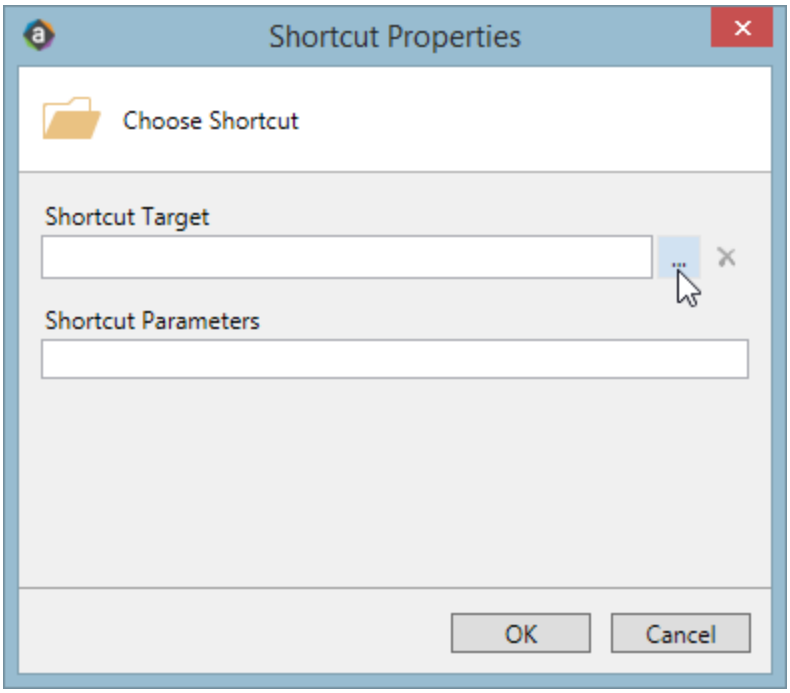
Configuring a Button component to manage attachments

To start off, add a Button component to the Axiom form canvas and then configure the button properties as desired. You will probably want the display text of the button to be something like "Attachments". You can then configure the **Command** for the component as follows:

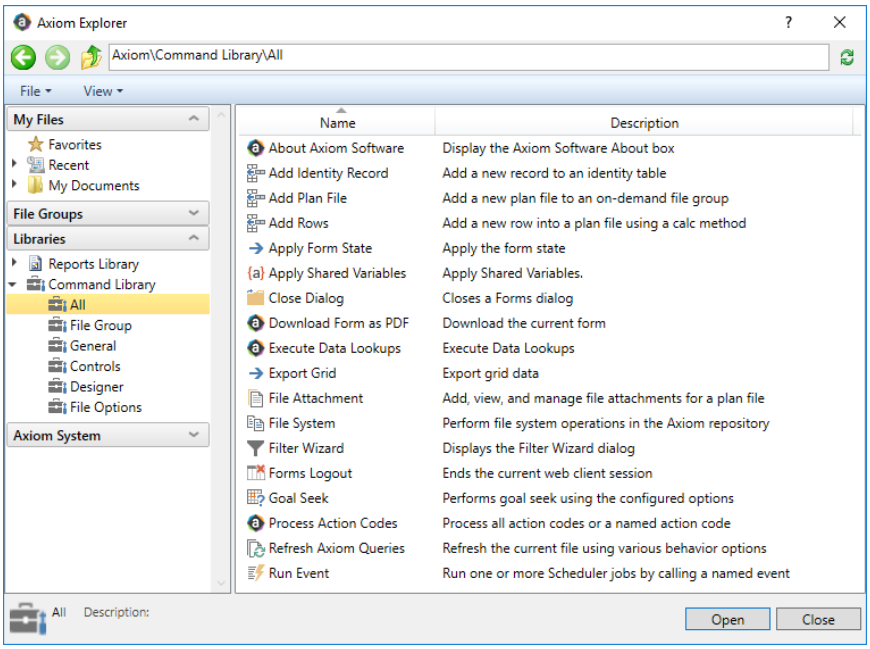
1. In the component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



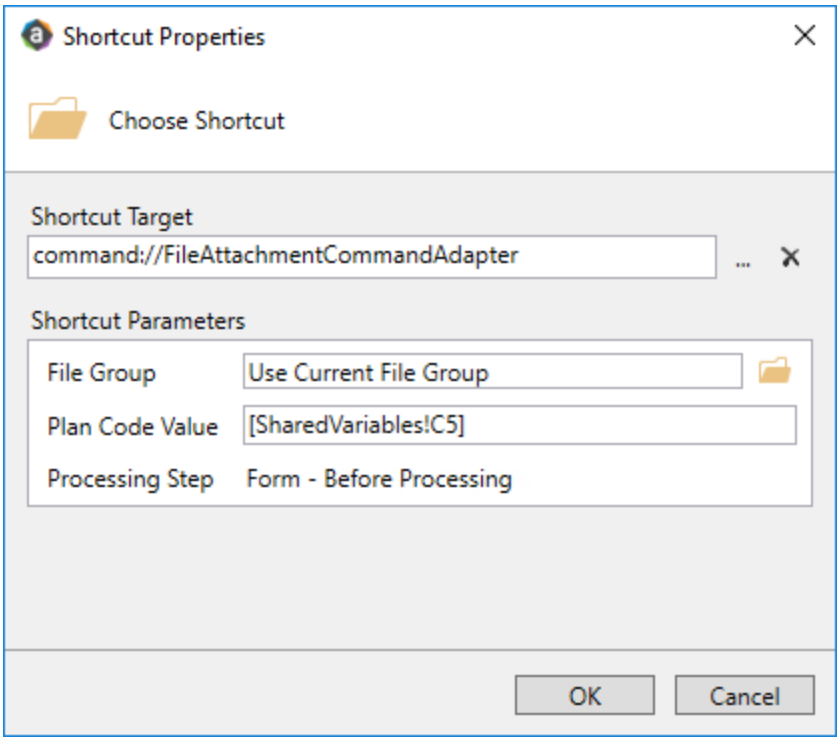
3. In the **Axiom Explorer** dialog, select **Command Library > All**, then select the **File Attachment** command. Click **Open**.



In the Shortcut Properties dialog, the File Attachment command is now listed as the shortcut target, and the relevant shortcut parameters are now available.

4. In the **Shortcut Parameters**, complete the following settings:

Item	Description
File Group	The file group for the command. Click the folder icon to select a file group. You can select any file group or file group alias. If you are using the command in a file that belongs to a file group, then an additional option is available at the top of the file group list: **Use Current File Group**. This option means that the command will always point to the file group where the file is currently located, even if you clone the file group or copy the file to a different file group. This is the recommended option when using this command in a file that belongs to a file group.
Plan Code Value	The plan file for which to open the Web Client File Attachments dialog. You can specify the plan code value directly, or use a bracketed cell reference to read it from a cell in the file. For example: `[SharedVariables!D12]`
Processing Step	Specifies when the command will be executed during the Axiom form update process. This option is set to **Form - Before Processing** and cannot be changed. Remember that the form update cycle does not occur when using this command—the command is executed before processing begins and then the update process is aborted.



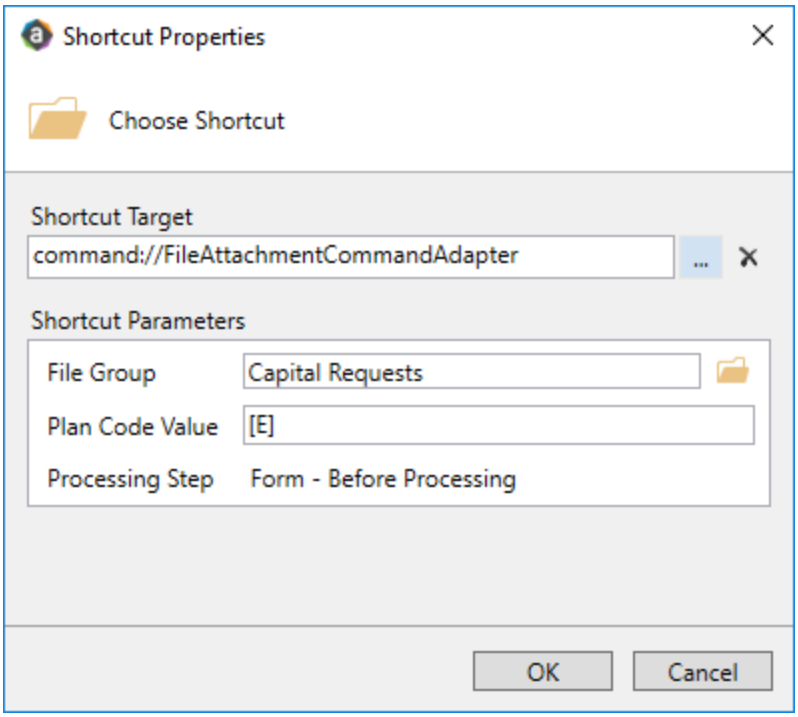
Example shortcut properties for File Attachment command

5. Once you have finished configuring the Shortcut Properties, click **OK** to close the dialog and return to the component properties.

Using a Button tag in a Formatted Grid component

Button tags in thematic Formatted Grid components can also be configured to run this command. In this case, use the Command parameter within the tag to assign the command to the button. The easiest way to do this is to use the Tag Editor dialog or the Data Source Assistant to create the tag and edit the tag parameters. When using these helper dialogs, you can select the command and configure the shortcut parameters using the same method described previously for the Button component.

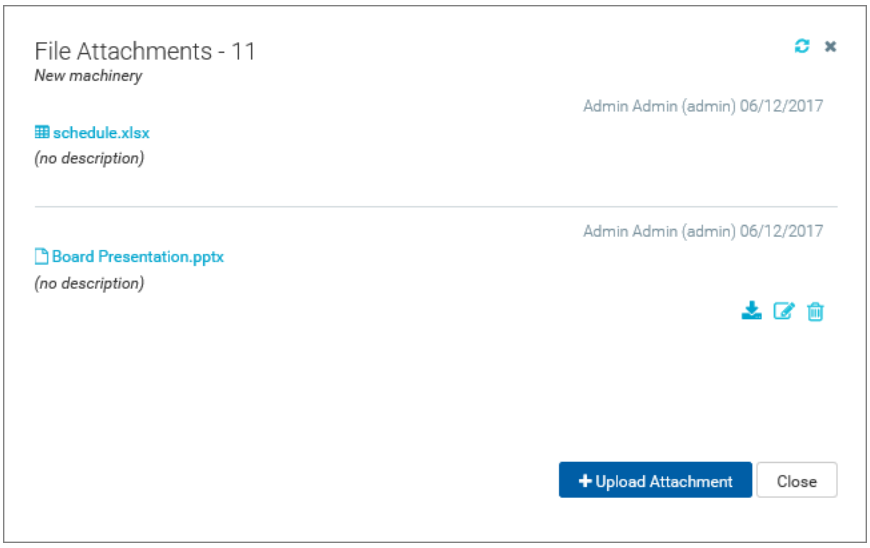
When using a Button tag, you can use either a bracketed cell reference or just a column letter in brackets within the **Plan Code Value** property. If using a column letter, such as `[J]`, then the plan code value will be read from the specified column on the current row of the grid.



Example shortcut properties using column-only syntax for Plan Code Value (Button tag)

User experience

When a user clicks the button with the File Attachment command, the File Attachment dialog opens:



Existing attachments are listed in "blocks" with the attachment name on the left and the actions on the right. To view the actions for a particular attachment, hover over the attachment block.

- Users with read / write access to the plan file can open (download) the attachment, edit attachment properties, and delete the attachment. These users can also upload new attachments using the Upload Attachment button.
- Users with read-only access to the plan file can open (download) the attachment. The other actions do not display in the dialog.

Using a button in a form to upload attachments

You can use a Button component within a form-enabled plan file to allow users to upload attachments associated with that plan file. This functionality can be used as a substitute for the Attachments panel in the Web Client Task Bar, or in conjunction with the panel.

NOTE: This feature is primarily available for backward-compatibility. Going forward, the File Attachments panel or the File Attachment command should be used instead to allow users to upload, view, and manage attachments in the Web Client.

Requirements and limitations

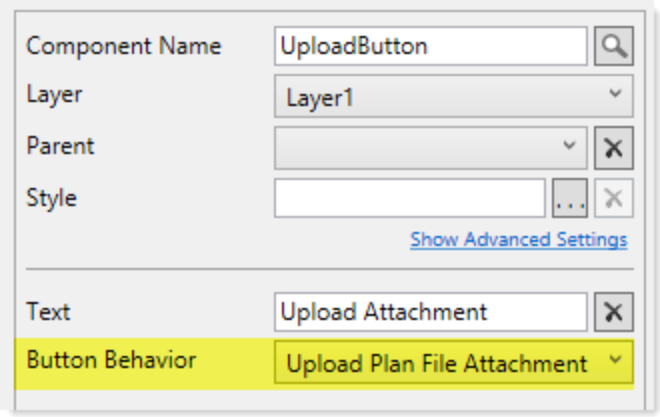
- You can allow users to upload new attachments for a plan file by using a Button component that is configured to use the **Upload Plan File Attachment** button behavior. This behavior option is only valid for use in form-enabled plan files, and only if the file group has enabled the plan file attachments feature. If you configure this behavior in an environment that does not support use of it, then the upload button will still operate but an error will occur when attempting to upload the attachment.

NOTE: This button behavior is only available for Button components. Button tags for Formatted Grid components do not support this behavior.

- When using the upload button, the normal button refresh process occurs after the upload message box is dismissed (whether the message reports success or failure). There is no way to disable this refresh.
- The upload button only allows uploading attachments. Attachments cannot be opened, edited, or deleted using the button.

Configuring a Button component to upload attachments

To allow Axiom form users to upload file attachments using a button, you must add a Button component to the form and set the **Button Behavior** to **Upload Plan File Attachment**. The button text should be set to "Upload Attachment" or something similar.



Example upload button configuration

This button should be placed in the template file, so that all plan files created from the template have access to the button.

User experience

The following is a summary of the button behavior within an Axiom form, so you have a better understanding of the user experience.

1. The user clicks the button when they want to upload a file attachment.
2. The **Choose File to Upload** dialog opens. The user browses to the file that they want to upload as an attachment.

NOTE: If the user is on a tablet, they are given the option to select an existing picture to upload or take a new picture to upload.

3. A message box shows the upload progress, and then shows when the upload is complete (or if it failed).
4. The Axiom form automatically refreshes after the message box is dismissed.

Hyperlinking to plan file attachments within a form

You can display a list of plan file attachments in an Axiom form, and provide hyperlinks to open those attachments. You can display this list in any Axiom form, not just within the plan file. For example, you could have a form-enabled report that shows all of a user's plan file attachments (across multiple plan files).

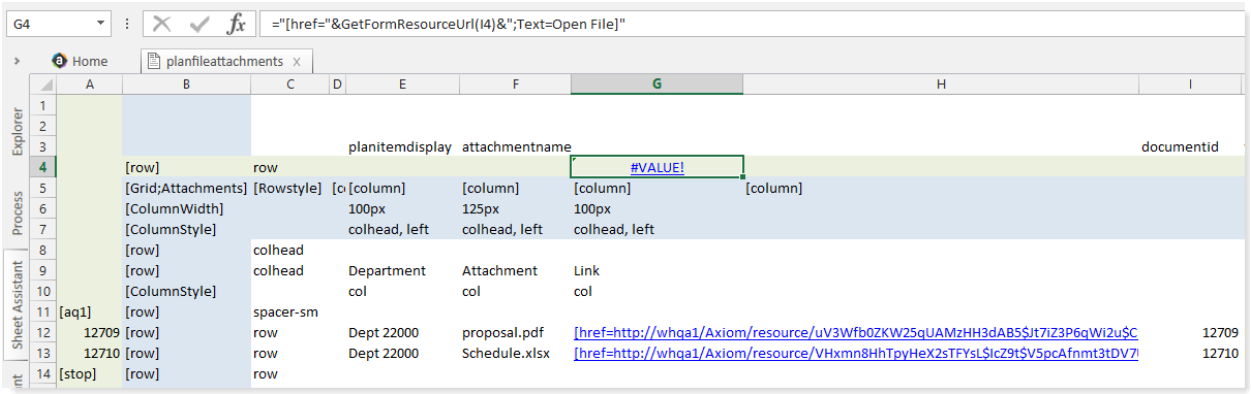
You can set up this list as follows:

1. In the source file for the Axiom form, create an Axiom query to the Axiom.PlanFileAttachments table to return the desired list of attachments and the desired properties. In most cases you will want to apply a data filter to the query so that it is limited to a certain file group and to certain plan files within that file group.
2. In the in-sheet calc method for the Axiom query, use the HREF content tag for Formatted Grid components to generate hyperlinks for the attachments returned by the query. You can manually create the HREF tag and use the GetFormResourceURL function to provide the URL to the attachment, or you can use the GetFormResourceLinkTag function to automatically generate the HREF tag for you. For more information, see Using hyperlinks in Formatted Grids.

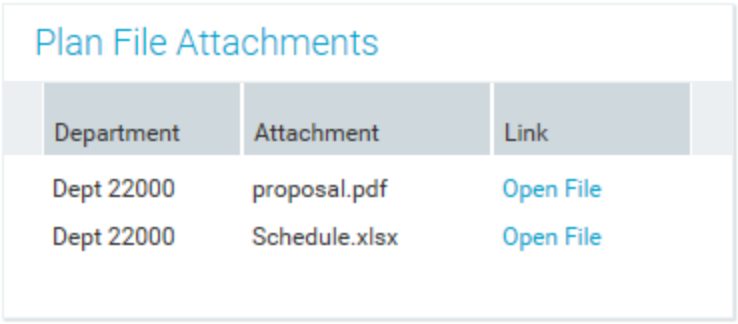
   NOTE: If you are manually creating the HREF tag, do not include the UseNewWindow parameter. This parameter should be omitted when creating a hyperlink to a plan file attachment.

3. Set up a Formatted Grid component on the Axiom form canvas, using the Axiom query results as a data source for this grid.

The following screenshot shows an example Axiom query to the Axiom.PlanFileAttachments table, with the data tagged as a data source for a formatted grid. The field definition for the query is row 3 and the in-sheet calc method is row 4. Note that the `[row]` tag in B4 will be ignored because it is above the Grid data source tag; it is being used in the calc method to tag the rows returned by the Axiom query.



In the rendered Axiom form, the Formatted Grid component would appear as follows. Users can click on the "Open file" hyperlinks to open the associated attachments.



Inserting calc methods in an Axiom form

If users are interacting with plan files as Axiom forms, then limited functionality is available to allow Axiom form users to insert new rows using calc methods. However, the "change calc method" functionality is not available in Axiom forms.

There are two ways to configure an Axiom form to allow end users to insert calc methods:

- You can set up a Button component to use the Add Rows command. The user clicks on the component in the Axiom form to insert the new row.
- You can set up a cell in a Formatted Grid component to trigger inserting rows. For spreadsheet-formatted grids, you can use the AddRows content tag. For thematic grids, you can use the Button content tag with the Add Rows command.

As normal, the user must have the **Allow Calc Method Insert** security permission for the file in order to perform this action.

Requirements and limitations

To use calc method insert functionality in an Axiom form, your template and calc methods must be set up as follows. This applies regardless of which method is used to allow insertion within the Axiom form.

- An InsertCMColumn must be defined, and the tag must have a defined header label. Dynamic insertion controls are not supported in Axiom forms.

  For example: `[InsertCMColumn;CMInsert]`

- An InsertCM tag must be placed on the row where you want to allow insertion. This tag must have a defined insertion point label, and must allow insertion. You can specify one or more calc methods or groups to allow for insertion, or you can leave those parameters blank to allow any calc method.

  For example: `[InsertCM;InsertNewRow;New Row]`

- In most cases, the calc method to be inserted should contain both a `[row]` tag and a `[save]` tag in the appropriate columns, so that the newly inserted row displays in the Formatted Grid component, and so that the data for the newly inserted row gets saved to the database. Remember that the source file itself is not saved when the Axiom form triggers a save, so the newly inserted row will *not* be retained in the source file, only its data can be saved.

  NOTE: Neither of these tags are strictly required if you have a use case where you want to insert a row that does not subsequently display in the Axiom form, or does not get saved to the database. For example, perhaps the data for the new row impacts another component, or the data gets summed and saved from a different row.

- Only one instance of the calc method can be inserted at a time within Axiom forms. The Axiom form environment does not provide an option to insert multiple instances of the calc method at a time. The MaxInsertCount parameter on the InsertCM tag will be ignored.
- If the calc method uses calc method variables, the user can only set the values of these variables once, when the calc method is originally inserted. If you need the user to be able to edit these values after insertion, then the target cells must be configured as editable (unlocked). Keep in mind that if you use a Related Column Value variable, the related value will not be automatically updated if you allow the user to edit the parent value after insertion.

For more information on setting up calc method controls—meaning the InsertCMColumn and InsertCM tags—see the File Group Administration Guide.

The Add Rows command is always executed on the form, at the After Updating Values processing step of the form update process. For more information on how processing steps fit in with the form update process, see Axiom form update process.

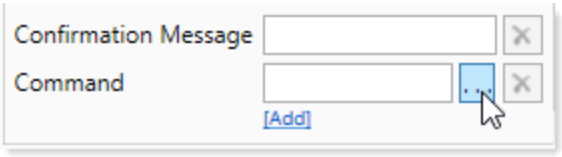
Setting up calc method insertion using a button

To allow users to insert new rows from the Axiom form, you can use a Button component that is configured to run the Add Rows command shortcut. This is the same command that is available for use in task panes.

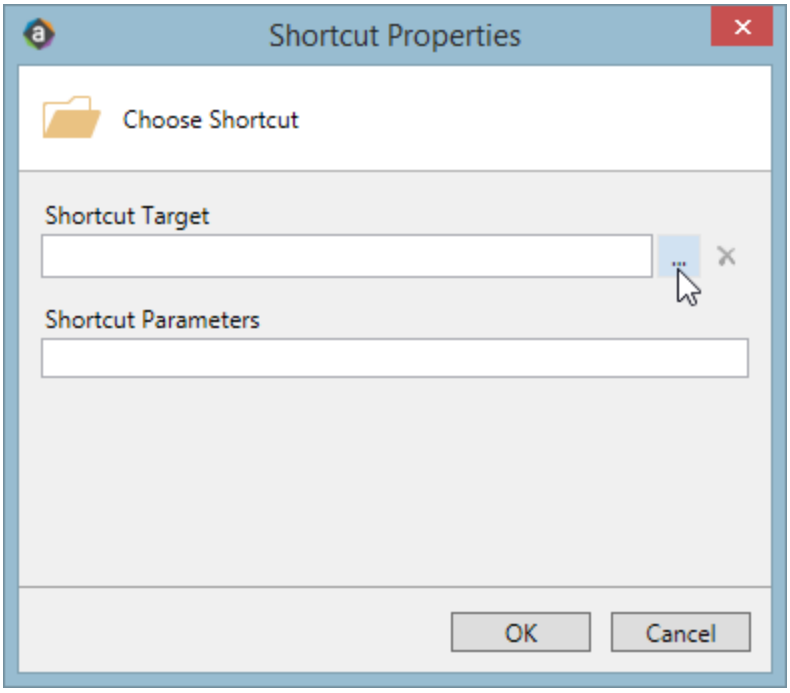
NOTE: In a thematic Formatted Grid component, you can use a Button content tag that is configured to run the Add Rows command. When creating the Button tag, the setup of the Add Rows command is the same as described in the following steps. For more information on creating a Button tag in a thematic grid, see Using buttons in Formatted Grids.

To start off, add the Button component to the Axiom form canvas and then configure the properties as desired. You will probably want the button text to be something like "Insert New Row". You can then configure the **Command** for the component as follows:

1. In the component properties, click the **...** button to the right of the **Command** box.



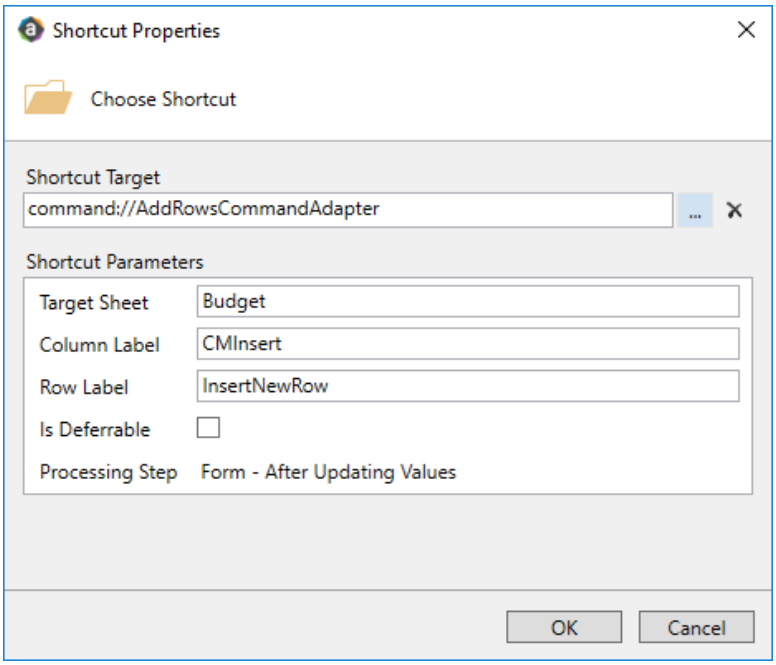
2. In the **Shortcut Properties** dialog, click the **...** button to the right of the **Shortcut Target** box.



3. In the **Axiom Explorer** dialog, select the **Command Library**, then select **Add Rows**, then click **Open**.

   The Add Rows command is now listed as the shortcut target, and the relevant shortcut parameters are now available. All three parameters are required.

4. Complete the shortcut parameters as follows, then click **OK** to close the Shortcut Properties dialog.
   - **Target Sheet**: Type the sheet name where the calc method is to be inserted.
   - **Column Label**: Type the header label defined in the InsertCMColumn tag.
   - **Row Label**: Type the insertion point label defined in the InsertCM tag. This is where the calc method will be inserted.



Example Shortcut Properties dialog

Using Is Deferrable

The **Is Deferrable** option on the Add Rows command is intended for cases where you have multiple Add Rows commands on a single button. If some insertions use calc method variables and others do not, you can defer the execution of the insertions without variables until all variable values have been selected. If the user cancels a variable selection and therefore cancels that insertion, the deferred insertions will also be canceled.

To do this, set up the button commands as follows:

- List the Add Rows commands that use variables before the commands that do not use variables, so that the insertions with variables are executed first.
- Enable **Is Deferrable** for all of the Add Rows commands that do not use variables, so that insertions without variables are deferred and dependent on the completion of the insertions with variables.

Whether an Add Rows command uses calc method variables depends on the calc methods allowed by the InsertCM tag for the specified **Row Label**. Deferred insertions must be assigned a single calc method that does not use variables, because deferred insertions must be able to be processed without user input.

Setting up calc method insertion using an AddRows tag

To allow users to insert new rows from the Axiom form, you can use content tags within a Formatted Grid component. The AddRows content tag will display as a clickable hyperlink when the file is viewed as an Axiom form. Users can then click on the hyperlink to add a new row.

NOTE: The AddRows content tag is only supported for use in spreadsheet-formatted grids, for backward-compatibility. Thematic grids should use the Button tag instead, and configure that tag to use the AddRows command.

The syntax for the AddRows tag is as follows:

`[AddRows; Sheet=`*`SheetName`*`; Column=`*`ColumnLabel`*`; Row=`*`RowLabel`*`; Text=`*`DisplayText`*`; Foreground=`*`color`*`; Background=`*`color`*`]`

Parameters can be listed in any order after the AddRows tag. You do *not* need to indicate omitted parameters with an "empty" semi-colon.

Parameter	Description
Sheet	The sheet name where the calc method is to be inserted.
Column	The header label defined in the InsertCMColumn tag.
Row	The insertion point label defined in the InsertCM tag. This is where the calc method will be inserted.
Text	The display text for the hyperlink that displays in the formatted grid. Users will click this hyperlink to insert the new row.
Foreground	Optional. The foreground color to use for rendering the cell contents (text, symbol, etc.). This parameter only applies to spreadsheet-formatted grids. It will be ignored by thematic grids. By default, the text uses the font color defined for the cell in the spreadsheet. If you want to override this formatting and specify a color in the tag itself, you can use the Foreground parameter. The advantage of this approach is that the color can be made dynamic using a formula. The color can be specified using either the color name (i.e. "yellow") or the hexadecimal code for the color (#FFFF00 for yellow). For example, see a list of colors here: http://www.w3.org/TR/css3-color/#svg-color. When using the Data Source Assistant / Tag Editor, you can click the arrow button to the right of the box to bring up the color selector. You can select from the displayed colors, or you can enter a RGB value or a hexadecimal code. The selected color will be inserted in the tag using its hexadecimal code. To clear the selected color, click the **Clear color** ✕ icon. Alternatively, you can use a bracketed cell reference to read the color from the referenced cell. This approach is useful if you want to dynamically determine the color, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.

Parameter	Description
Background	Optional. The background color to use for the cell in the grid. This parameter only applies to spreadsheet-formatted grids. It will be ignored by thematic grids. By default, the grid cell uses the fill color defined for the cell in the spreadsheet. If you want to override this formatting and specify a color in the tag itself, you can use the Background parameter. The advantage of this approach is that the color can be made dynamic using a formula. The color can be specified using either the color name (i.e. "yellow") or the hexadecimal code for the color (#FFFF00 for yellow). For example, see a list of colors here: http://www.w3.org/TR/css3-color/#svg-color. When using the Data Source Assistant / Tag Editor, you can click the arrow button to the right of the box to bring up the color selector. You can select from the displayed colors, or you can enter a RGB value or a hexadecimal code. The selected color will be inserted in the tag using its hexadecimal code. To clear the selected color, click the **Clear color** ✕ icon. Alternatively, you can use a bracketed cell reference to read the color from the referenced cell. This approach is useful if you want to dynamically determine the color, because then the formula can be in the referenced cell instead needing to construct the tag using a formula. For more information, see Referencing cells in content tag parameters.

This tag works in the same way as the Add Rows command in the Command Library. When the user clicks on the hyperlink, Axiom will create an Add Rows command string using the parameters listed here, and then perform the Add Rows action.

To create the tag, you can manually type it within a cell, or you can use the Data Source Assistant / Tag Editor. For more information, see Creating and editing content tags in Formatted Grids.

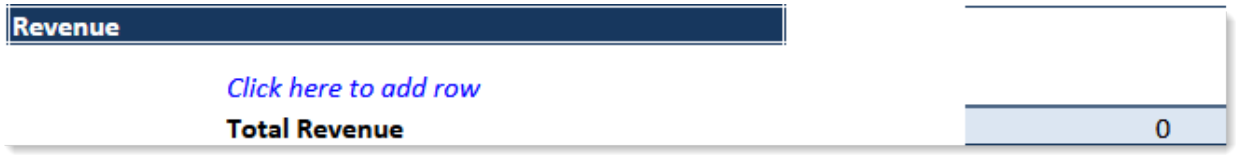
For example:

```
[AddRows;Sheet=Budget;Column=CMInsert;Row=InsertNewRow;Text=Click here to add row]
```

When a user views the file as an Axiom form, this will display as a hyperlink that says "Click here to add row". When the user clicks on the hyperlink, a calc method will be inserted on the specified sheet, using the specified column and row labels to determine the location.

The following screenshot shows an example of how the AddRows tag might look in the spreadsheet:

When this file is viewed as an Axiom form, the tag will be rendered in the form as a clickable hyperlink:



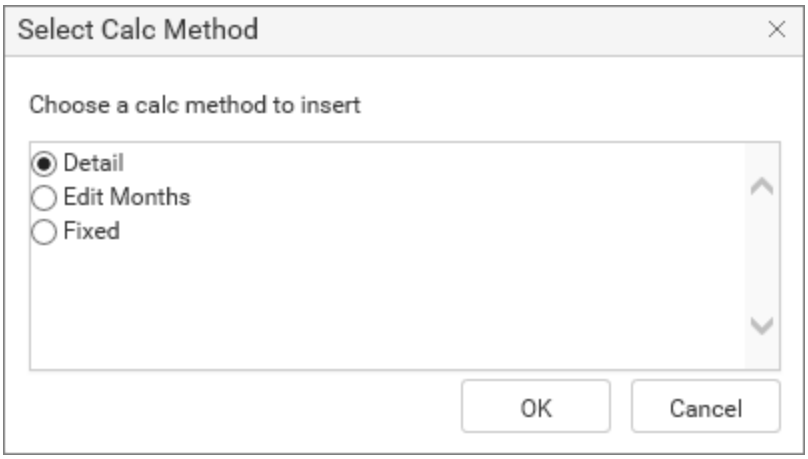
NOTES:

- Use of this tag to perform an Add Rows action automatically triggers a refresh of the Axiom form. First, updated values will be submitted from the form to the source file. Then, the Add Rows action will occur in the source file. Lastly, the source file will be refreshed and then the Axiom form will be updated.
- Remember that the insert location is not necessarily the row that contains the AddRows tag; it is whatever location is specified by the Sheet / Column / Row parameters (which can be the current row but does not have to be). In other words, this tag does *not* behave like the GetCalcMethod function for Axiom files (which always inserts on the current row).

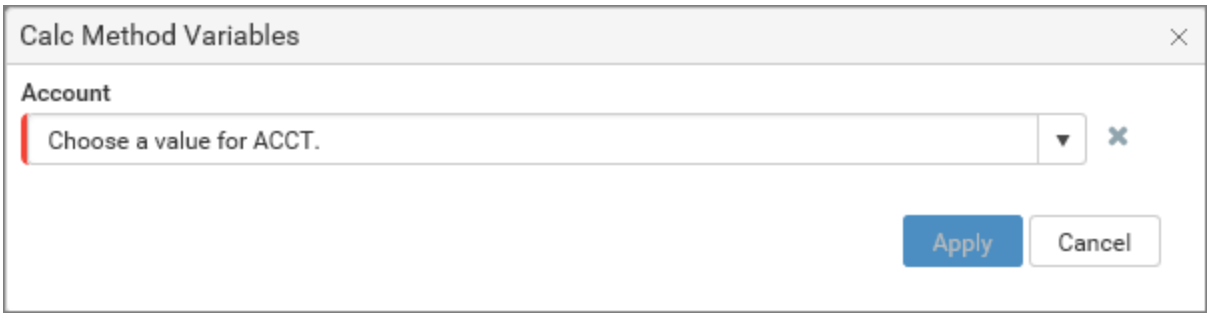
Add Rows behavior

Within the Axiom form, the user can click the button or the hyperlink to perform the Add Rows command. Within the source file, the command identifies the designated sheet, column, and row, and then inserts the calc method as specified in the InsertCM tag.

If multiple calc methods are allowed for insertion, then the user is first prompted to select which calc method to insert.



If the calc method has calc method variables, then the user is prompted to specify values for those variables before the calc method is inserted.



Assuming the inserted row is configured to display in the formatted grid, the new row will display in the Axiom form after the form update process is complete. The user can then complete any inputs associated with the new row.

Working with plan file process tasks in Axiom forms and the Web Client

Axiom Software provides full functionality to view and complete plan file process tasks using Axiom forms and the Web Client environment. This topic provides a summary of the various features that are available, so that you can determine which features are most appropriate for use in your system.

Completing the current process task within a form-enabled plan file

When the current step owner is working on a form-enabled plan file, you may want to allow that user to complete the task from within the plan file. You can do this by configuring Button components in the plan file template with the Submit Process and Reject Process button behaviors. The current step owner can click these buttons to complete the current process task. For more information, see Completing the current process task in a form-enabled plan file.

Completing process tasks using the Web Client

Web Client users can access the Process Tasks web page to review all of their currently active process tasks for a particular plan file process, and then complete those tasks individually or in batch. This page is accessible by providing users with a URL to the page (either directly or within an Axiom form). For more information, see Completing process tasks using the Process Tasks page.

Viewing process status using the Web Client

Web Client users can access the Process Directory web page to review the process status of all plan files that they have access to. If the user is the current step owner of a plan file (or an administrator or a process owner), the user can also complete the process task from this web page. This page is accessible from the Process Summary component in an Axiom form, or by providing users with a URL to the page (either directly or within an Axiom form).

Web Client users can also access the Process Routing web page to review the process status and details of individual plan files that they have access to. This page provides full process details for the plan file, and can also display summary information about the plan file itself and its data. If the user is the current step owner of the plan file (or an administrator or a process owner), the user can also complete the process task from this web page. This page is accessible from the Process Tasks web page and the Process Directory web page. You can also provide users with a URL to the page (either directly or within an Axiom form).

For more information, see Viewing process status using process web pages.

Viewing process status within any Axiom form

When users access their form home page (or a similar landing page), you may want to provide those users with an at-a-glance summary of their current process tasks. You can do this by placing a Process Summary component on the Axiom form. This component is intended to remind users of current tasks, alert them about new and important tasks, and provide navigation to process web pages so that tasks can be completed. For more information, see Process Summary component.

Completing the current process task in a form-enabled plan file

Form-enabled plan files can be part of a plan file process. If you want to allow the current step owner to complete the process task from within the plan file, you can configure a Button component in the template to provide this functionality.

This provides similar functionality to the Desktop Client option that allows users to complete the current task when they save the plan file. In the forms environment, the user is not prompted on save, but instead must click a designated button on the form if they want to complete the task.

Requirements and limitations

The ability to complete the current process task is available for form-enabled plan files as follows:

- You can use a Button component in the form to allow users to complete the current process task. There are two button behavior options for plan file processes:
  - Submit Process, which is equivalent to **Mark step as complete** for Edit Plan File tasks and **Approve** for Approval tasks
  - Reject Process, which is equivalent to **Reject** for Approval tasks

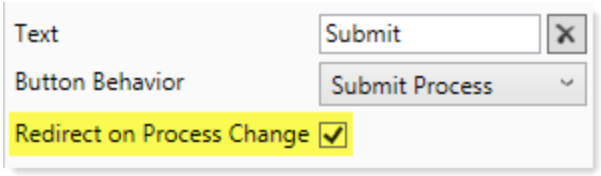
  These button behaviors are only valid for use within the form-enabled plan file, to complete the current task. The buttons are not available for use in any other form context (for example, you *cannot* create a form task pane for use with a regular plan file and use the buttons in the task pane). If the plan file is a composite form, the buttons must be located in the parent template, not in one of the child utility forms.
- When a user clicks one of these buttons, a form update and a save-to-database always occur before the Process dialog displays. The **Save on Submit** option for the button does not apply and is ignored in this circumstance.

Setting up process buttons for a form-enabled plan file

To allow users to complete the current process task in a form-enabled plan file, you must add two Button components to the form. This is done at the template level so that all plan files built from the template have access to these buttons. The **Button Behavior** for one button should be set to **Submit Process**, and the button behavior for the other button should be set to **Reject Process**.

When either of these behaviors are selected, one additional component property becomes available for the button: **Redirect on Process Change.** This setting determines what occurs after a user has used the button to complete the current process task:

- By default this option is disabled, which means the user is returned to the plan file after completing the task.
- If enabled, the user is redirected to the Process Routing page for the plan file after the current process task is completed.



NOTES:

- You should not enable this option if you have configured the process so that the Process Routing page is not available to end users (the option **Make routing page visible to anyone with read access to the plan file** is disabled on the plan file process definition). However, by default, end users can see the routing page.
- This option has no effect if the form-enabled plan file is opened as a web tab in the Desktop Client. In that case, the user is always returned to the plan file.

Additionally, you may want to use a formula to determine the label for the Submit Process button, so that it reads something like "Submit plan file" for an edit task and "Approve plan file" for a review task (the GetProcessInfo function can be used for this purpose). The label for the Reject Process button should read something like "Reject plan file".

Button behavior for end users

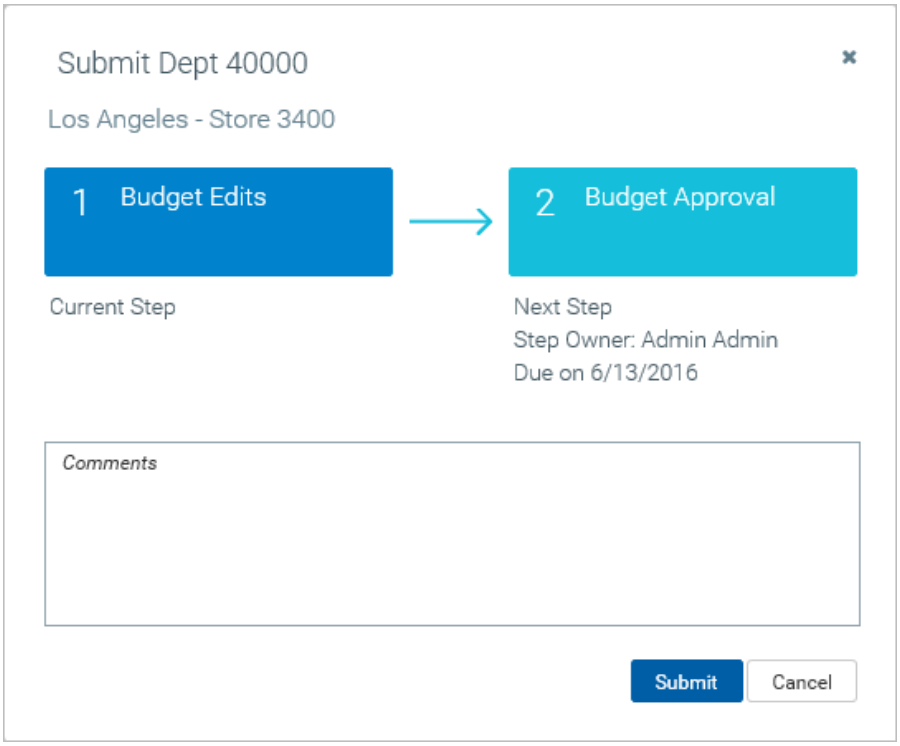
The buttons are active in the Axiom form as follows:

- If no plan file process is currently active for the plan file, or if the user is not the current step owner, then the buttons are hidden in the plan file.
- If the user is the current step owner of an edit step, then only the Submit Process button is visible and active. The Reject Process button is hidden.
- If the user is the current step owner of an approval step, then both buttons are visible and active.

The following is a summary of the button behavior within an Axiom form, so that you have a better understanding of the user experience.

1. The user clicks one of the process buttons to complete the process task for the current plan file.
2. An update is performed for the Axiom form, including a save-to-database. This process is automatic and cannot be disabled in the button configuration.

   If a save-to-database cannot be performed for the current step (for example, if it is an approval step without edit rights), then the save-to-database will not occur and no error will display.
3. The **Process Action** dialog displays. The user can enter a comment for the step. In the following example, the user is submitting the plan file to complete an edit task.



4. If the user clicks **OK**, the dialog is dismissed and the process task is completed. What happens at this point depends on whether **Redirect on Process Change** is enabled for the button:
   - If enabled, the plan file is closed and the user is automatically redirected to the Process Routing page for the plan file, where they can see that the plan file has been advanced to the next stage (or rejected backward if applicable).
   - If disabled, the user is returned to the plan file and no further feedback is provided to the user.

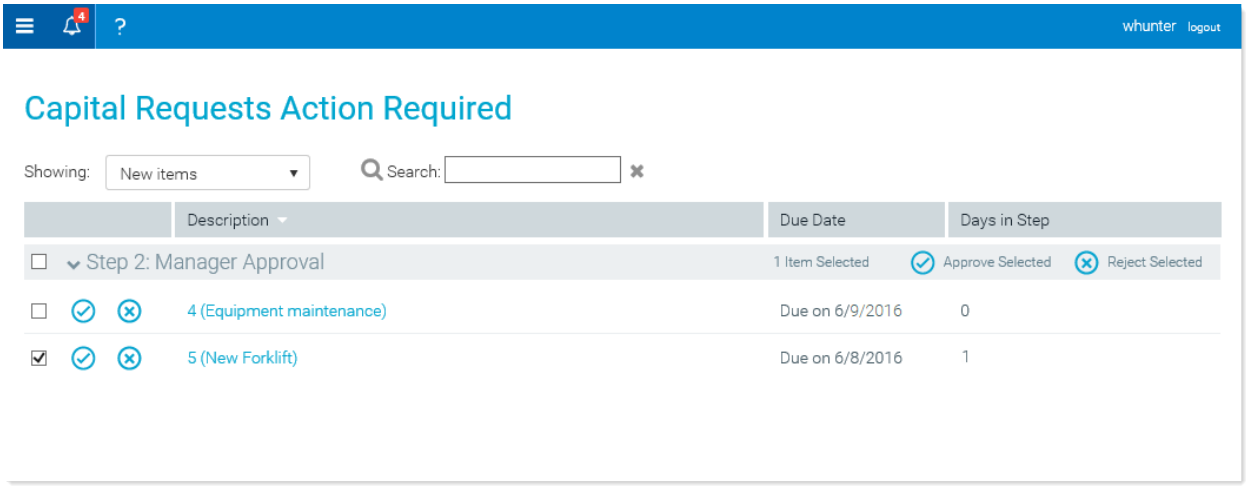
     At this point, the user is no longer the step owner of the plan file (unless they are also the owner for the next or prior step) and therefore they can no longer save data unless their security permissions explicitly permit it.

   If the user clicks **Cancel**, no process change occurs and the user is returned to the plan file.

NOTE: If **Save and validate plan file before advancing to next step** is enabled for the current step, and **Process Plan Files with Utilities** is enabled for the file group, then utility processing is performed for the plan file before the step is completed. If the utility processing fails, the step is not completed. If the utility processing succeeds, the step is completed as normal.

Completing process tasks using the Process Tasks page

Using the Process Tasks page of the Web Client, users can view all of their current process tasks, and can complete tasks individually or in batch. The following screenshot shows an example of this page.



The primary use case for this page is when plan files are form-enabled, and the main client for end users is the Web Client. The page can also be used with spreadsheet plan files, but in this case you must consider the user experience between the Web Client and the Desktop Client. For example, if users normally access the Web Client and only use the Desktop Client when they need to open a spreadsheet plan file, then it makes sense to provide users with access to this page so that users can complete tasks without needing to launch the Desktop Client. However, if the Desktop Client is the main client for end users, then it may be confusing for users to be directed to the Web Client to complete their process tasks.

Accessing the page

There is no built-in way to access this page; you must manually create the URL. In most cases, you will generate the URL and then include it in a home page (or a similar landing page), so that users can simply click a link to be taken to the page.

To access this page, use the following URL:

`<baseURLtoAxiom>/process/processID/user?login=loginname`

The login parameter can be omitted, and the page will show the tasks for the currently logged in user. If a login name is specified, then Axiom Software verifies that the current user is the specified user, or that the current user is an administrator or an owner of the process.

For example, if the process ID is 5988 and you want the page to show tasks for the current user, the URL would look as follows:

```
https://CustomerName.axiom.cloud/process/5988/user
```

If you want the page to show tasks for user jdoe, then the URL would look as follows

```
https://CustomerName.axiom.cloud/process/5988/user?login=jdoe
```

In this example, the URL will only work for user jdoe, an administrator, or a process owner. Any other user will see an error if they attempt to use the link.

You can find the ID for a process by using the GetProcessInfo function, or by hovering over the process definition in the Explorer task pane.

Viewing process status using process web pages

The Web Client provides two built-in web pages to view the process status of plan files:

- **Process Directory**: Using this page, users can view the process status for all plan files they have access to within a file group.
- **Process Routing**: Using this page, users can view the process status and details for an individual plan file.

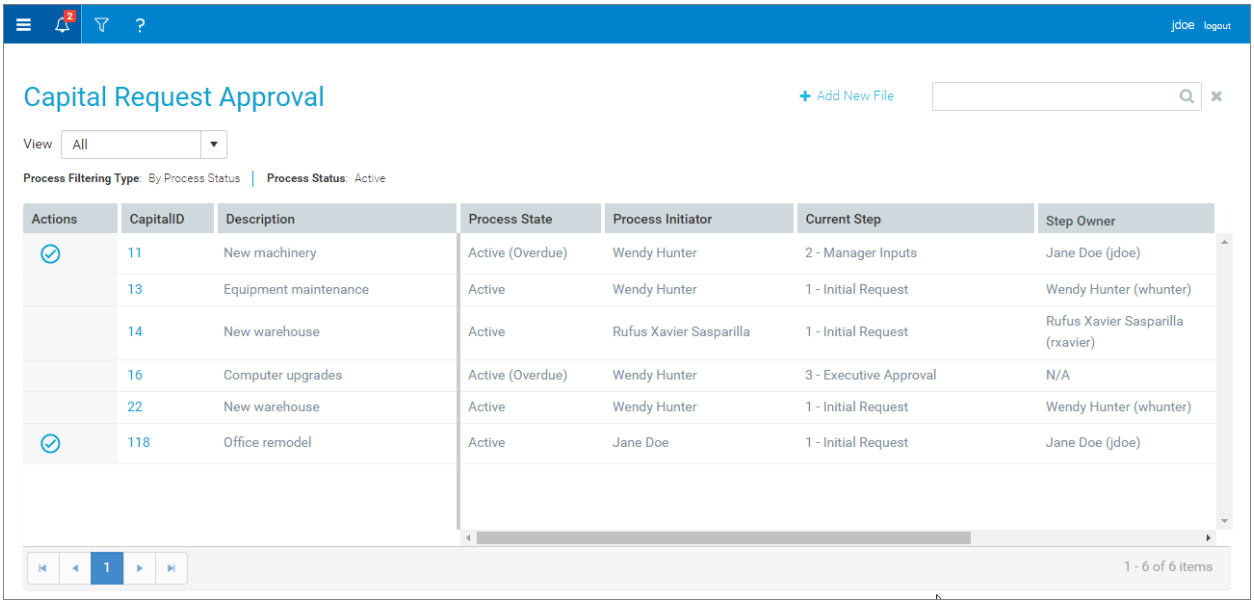
These web pages are primarily intended to be used when plan files are form-enabled, and the main client for end users is the Web Client. The pages can also be used with spreadsheet plan files, but in this case you must consider the user experience between the Web Client and the Desktop Client. For example, if users normally access the Web Client and only use the Desktop Client when they need to open a spreadsheet plan file, then it makes sense to provide users with access to the pages so that users can review process status without needing to launch the Desktop Client. However, if the Desktop Client is the main client for end users, then it may be confusing for users to be directed to the Web Client for this information.

Using the Process Directory page

Using the Process Directory page of the Web Client, users can view the process status of all plan files that they have access to.

- You can customize this page to specify which columns are shown, the initial sort level of the list, and other formatting options.
- To open the Process Routing page for a particular plan file, users can click the hyperlink in a designated column. In the following example screenshot, the CapitalID column is the designated hyperlink column.

- Users who are step owners of one or more plan files can complete the associated process tasks. Administrators and process owners can also complete tasks from this page.
- Users can use the search box at the top to locate a particular plan file. You can configure which columns are included in the search.
- Using the View options, users can see all of the plan files they have access to, or just the plan files for their current process tasks. If the file group is an on-demand file group, users can also see all of the plan files where they are the process initiator.
- Using the **Filters** panel, users can filter the page by process status, step status, and current owner. You can also optionally provide users with a set of predefined options to filter the list by plan code groupings, using refresh variables.



Example Process Directory page

The process details shown on this page are from the designated plan file process for the file group, as defined in the file group properties.

If the file group is an on-demand file group, users can also create new on-demand plan files by clicking the plus button in the top right-hand corner. This button uses the Add File Message text, and it only displays if the file group has a designated Add File Form (both as defined in the file group properties). Clicking the button launches the form. This is equivalent to the functionality in the Open Plan Files dialog to create a new on-demand plan file.

For more information on configuring this page, see the *File Group Administration Guide*.

Accessing the page

There are two ways to access this page:

- Users can click the totals links in the Process Summary component to be taken to this page. This is the only "built-in" way for users to navigate to the page.
- Alternatively, you can provide users with a hyperlink to the page, using the following syntax:

  `<baseURLtoAxiom>/filegroups/filegroupID/process`

For example, if the file group ID is 50, the URL would look as follows:

`https://CustomerName.axiom.cloud/filegroups/50/process`

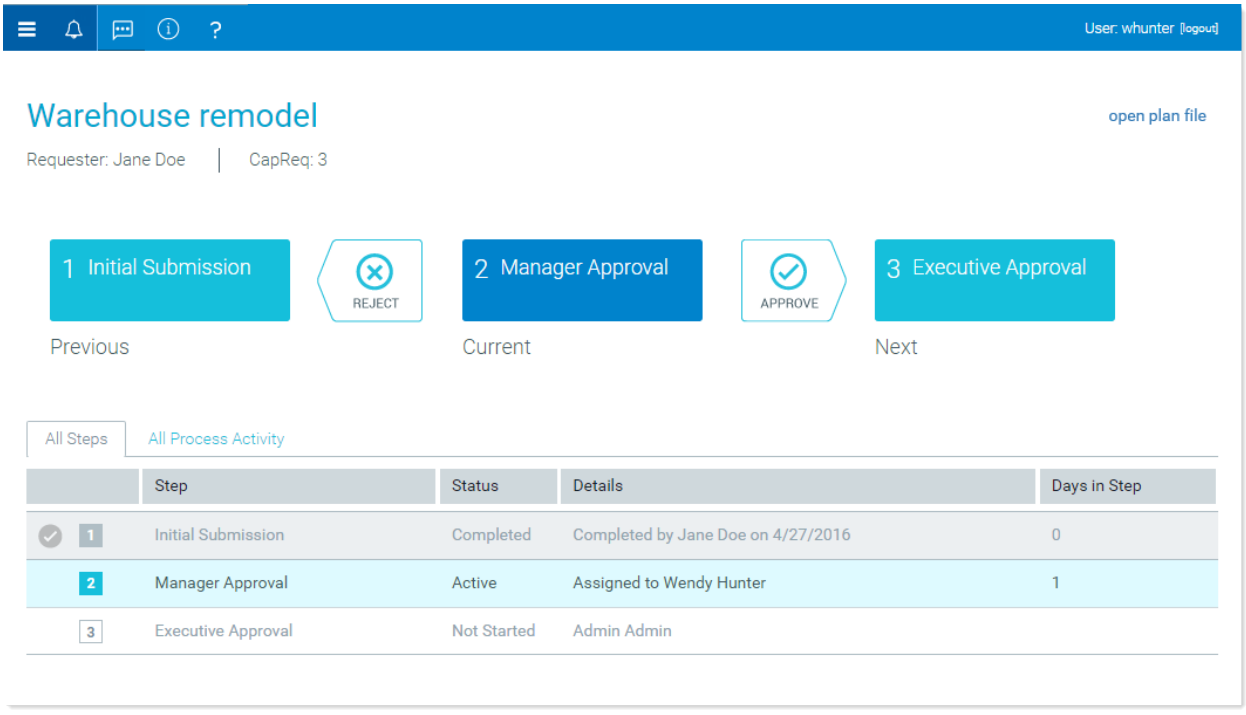
You can use the GetFileGroupID function to return the ID for a file group. To make the link dynamic, you can use a file group alias in the function. For example:

```
=GetFileGroupID("Current Budget")
```

Where Current Budget is the name of a file group alias that always points to the file group for the current budget cycle.

Using the Process Routing page

Using the Process Routing page of the Web Client, users can view the process progression of a particular plan file and open that plan file. If the user is also the current step owner, they can complete the current task from this page as well.



The availability of this page depends on the following setting in the plan file process definition: **Make routing page visible to anyone with read access to the plan file**. If enabled, then any user with at least read-only access to the plan file can access this page and view the process progression and activity details for the plan file. If disabled, only administrators and process owners can access this page.

If the user is not the current step owner, or an administrator, or a process owner, then the Process Routing page is for information only. The user can still see the current status of the plan file at the top of the page, but the user cannot complete the task.

Accessing the page

There are several ways to access the Process Routing page:

- Users can click any of the individual task links in the Process Summary component or in the Process Tasks page to be taken to the Process Routing page.
- Users can click on any hyperlinked column in the Process Directory page (see previous section).
- Alternatively, you can provide users with a hyperlink to the page, using the following syntax:

  `<baseURLtoAxiom>/process/processID/planfile?planvalue=code`

  The code is the plan file's dimension value from the key column of the plan code table. For example, if the plan code table is Dept, then the codes are department codes.

For example, if the process ID is 5988 and the plan file is for Dept 42000, the URL would look as follows:

`https://CustomerName.axiom.cloud/process/5988/planfile?planvalue=42000`

You can find the ID for a process by using the GetProcessInfo function, or by hovering over the process definition in the Explorer task pane.

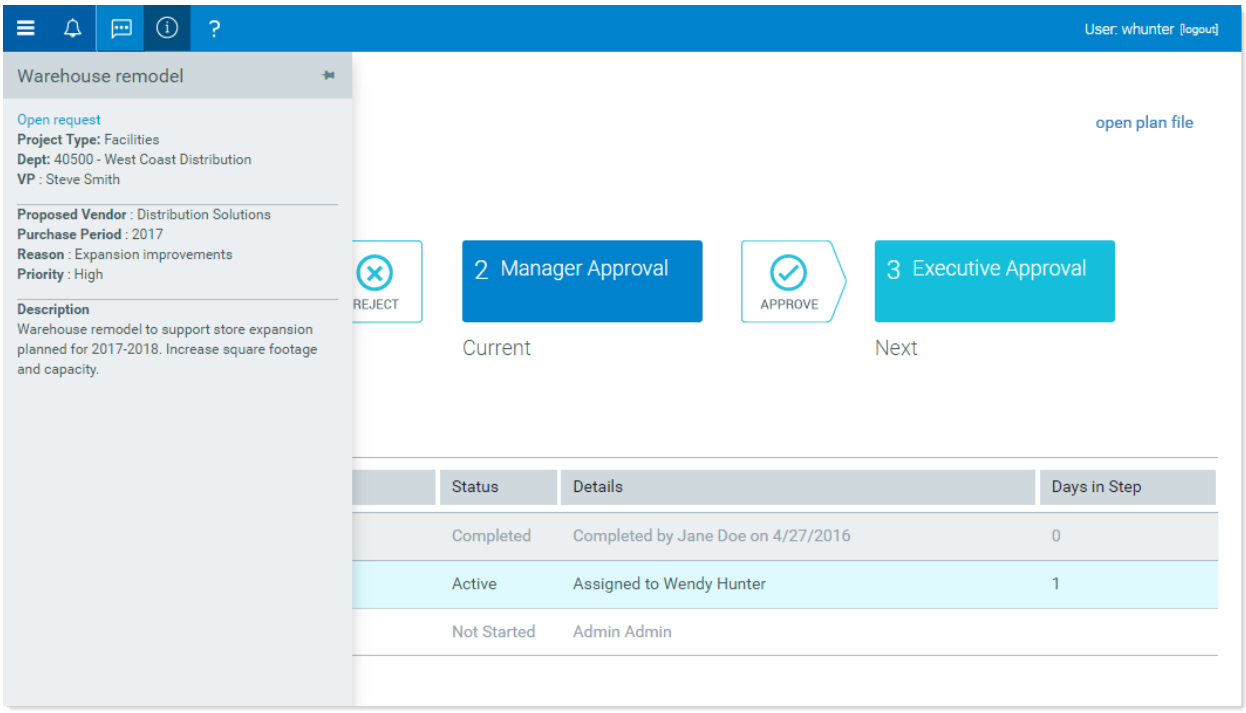
You may want to provide users with hyperlinks to this page so that they can continue to view the process details even when they are not the current step owner. For example, if the process is for capital requests, you may want the initial requester to be able to view the routing details for the plan file for the duration of the process, so that they can see the current status of their request.

To do this, you could display hyperlinks to each user within a Formatted Grid component as follows:

- Use an Axiom query to bring in the plan codes for which the current user is the initial requester.
- Use a formula in the query's in-sheet calc method to build up the necessary URL for each plan file.
- Use another formula in the query's in-sheet calc method to wrap the resulting URL in an HREF content tag, to display the URL as a clickable hyperlink in the form.

Information panel

You can also optionally configure an information panel that users can access from this page. This customizable panel can display details about the plan file, to help users decide if they are ready to complete the process task for the plan file. For more information on how to configure this panel in the plan file process definition, see the *File Group Administration Guide*.



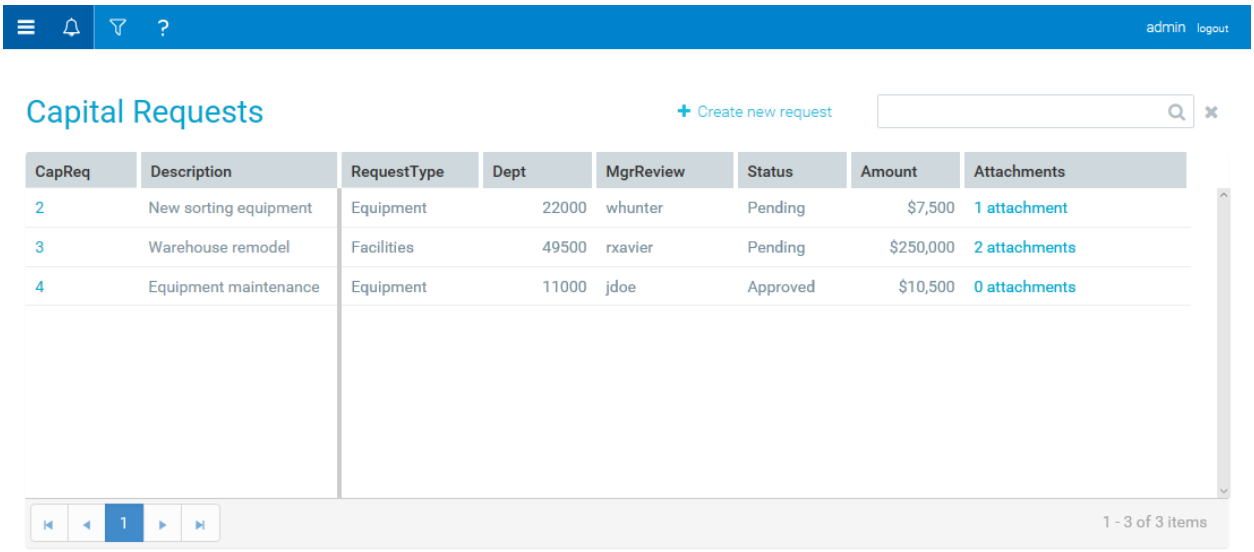
Example information panel

Using the Plan File Directory page

The Plan File Directory web page is available in the Web Client and serves a similar purpose as the **Open Plan Files** dialog in the Desktop Client. You can provide a link to this page within your Axiom forms to provide users with an easy way of opening their available plan files in the web.

The Plan File Directory page shows a list of plan files that the user has permission to access, for a particular file group.

- You can customize this page to specify which columns are shown, the initial sort level of the list, and other formatting options.
- To open a plan file, users click the hyperlink in a designated column. In the following example screenshot, the CapReq column is the designated hyperlink column.
- Users can use the search box at the top to locate a particular plan file. You can configure which columns are included in the search.
- If plan file attachments are enabled for the file group, users can view and manage attachments from the directory using the optional **Attachments** column.
- You can also optionally provide users with a set of predefined options to filter the list, using refresh variables and the **Filters** panel.



Example Plan File Directory page

If the file group is an on-demand file group, users can also create new on-demand plan files by clicking the plus button in the top right-hand corner. This button uses the Add File Message text, and it only displays if the file group has a designated Add File Form (both as defined in the file group properties). Clicking the button launches the form. This is equivalent to the functionality in the Open Plan Files dialog to create a new on-demand plan file.

The Plan File Directory page is primarily intended for use in cases where plan files are form-enabled, and the end users' primary client is the Web Client. However, the page can also be used as an alternate means to open spreadsheet plan files.

For more information on customizing this page, see the *File Group Administration Guide*.

Creating a link to the Plan File Directory page

There is no built-in way to navigate to the Plan File Directory page for a file group. If you want to use the page, you must manually generate the necessary URL and provide a hyperlink to your users. For example, you might have a form-enabled home page that contains hyperlinks to the currently active file groups in your system, so that your users can access their plan files using the Plan File Directory page.

To create a link to the Plan File Directory page for a file group, use the following syntax:

```
<baseURLtoAxiom>/FileGroups/FileGroupID/Directory
```

For example, if the file group ID is 50, the URL would look as follows:

```
https://CustomerName.axiom.cloud/FileGroups/50/Directory
```

You can use the GetFileGroupID function to return the ID for a file group. To make the link dynamic, you can use a file group alias in the function. For example:

```
=GetFileGroupID("Current Budget")
```

Where Current Budget is the name of a file group alias that always points to the file group for the current budget cycle.

Using Other Features in Axiom Forms

This section explains how to use various special features for Axiom forms.

Defining refresh variables for the Web Client Filters panel

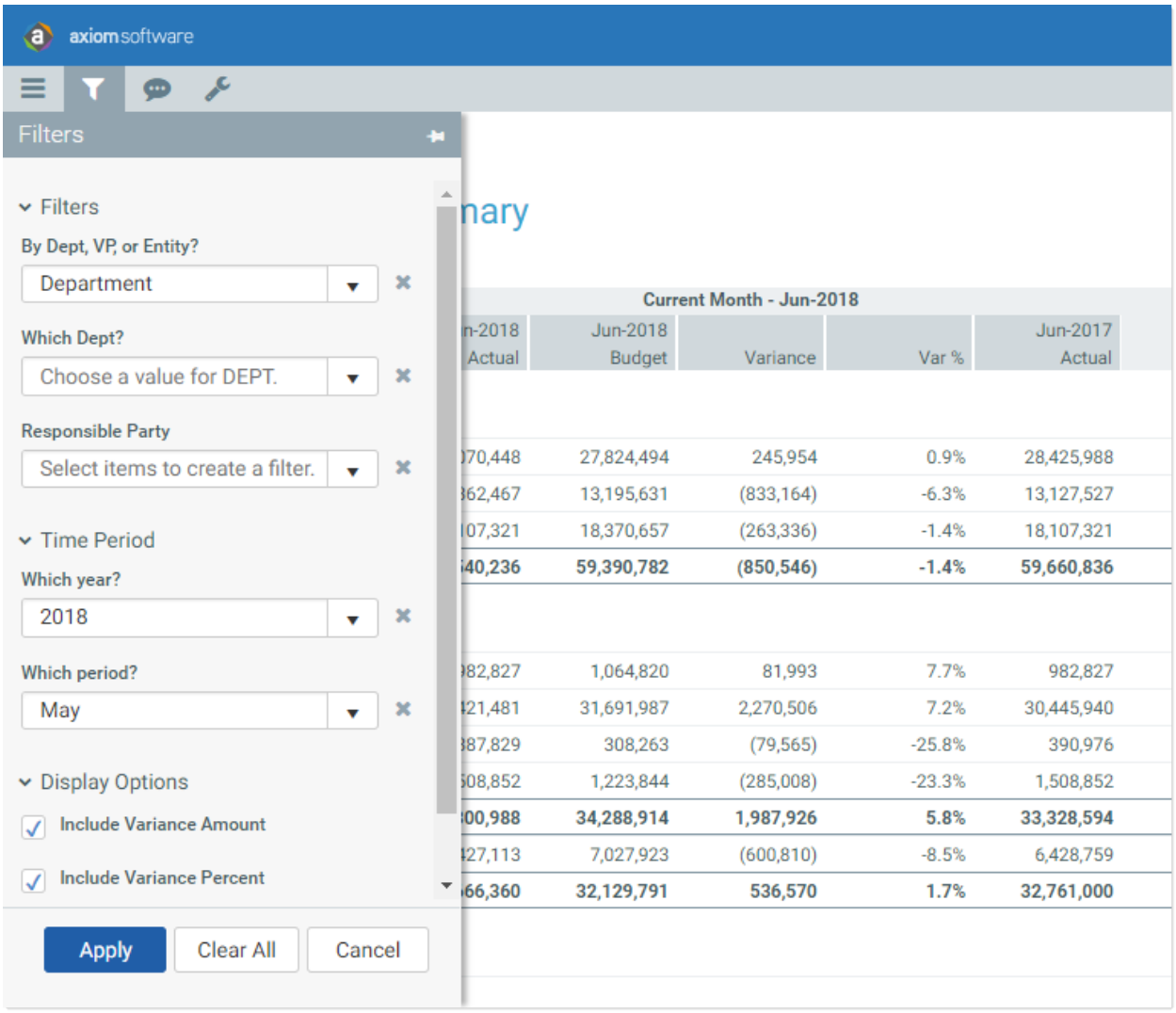
Refresh variables can be used in form-enabled files, to filter the data shown in an Axiom form. Users can select values for these variables and trigger a data refresh using the Filters panel in the Web Client. The data displayed in the form can then change based on the user's selections for the refresh variables.

The ability to use refresh variables provides a "built-in" option to allow users to change the data coming into an Axiom form. Instead of needing to manually create form components to gather user inputs, you can define refresh variables just like you would for a spreadsheet Axiom file. When the user clicks the filter icon in the Web Client Task Bar to indicate that they want to filter the data in the form, the refresh variables are automatically rendered in the Filters panel.

The basic operation of refresh variables in the Web Client works as follows:

- You create a form that displays data in some way, such as in a formatted grid and/or a chart.
- You define refresh variables in the form source file, and configure the data queries in the file so that they change depending on the user's inputs for those variables. For example, you might prompt users to select a business region, and then filter the data queries so that they return data for the selected region.
- When users view the form, they can use the built-in Filters panel to complete the inputs for the refresh variables and trigger a data refresh.
- The form is then updated to show the latest data based on their refresh variable selections.

If refresh variables are defined in the form source file, then form users can click the filter button to open the Filters panel. The Filters panel expands from the left side of the screen, and displays the refresh variables. The user selects values for the variables, and then clicks the **Apply** button to trigger the query refresh. The panel then retracts and the form is updated for the results of the query.



Example Filters panel

The user can optionally pin the Filters panel open by clicking the pin icon in the panel header. In this case, the filter panel no longer displays over the form; instead, it pushes the form to the side so that the user can see both the Filter panel and the entire form.

NOTE: The Web Client Container must be enabled for a form in order to use refresh variables. The container is enabled by default for all new forms.

Defining the refresh variables

To define refresh variables for use in a form, add a RefreshVariables data source to the form-enabled source file. The process to add the data source and define the individual variables is exactly the same as for use with a spreadsheet Axiom file. For more information on defining refresh variables, see the *Axiom File Setup Guide*.

Axiom forms support all of the refresh variable types and options, with the following exceptions:

- Grid variables should only be used when multi-select is enabled for the variable. If multi-select is not enabled for the variable, then it will display and behave like a ComboBox variable, so it is recommended to use a ComboBox variable instead.

  NOTE: The web version of the grid dialog does not display using separate columns, so the list is not sortable by column.

- The **Refresh Forms Run Behavior** setting on the Control Sheet does not apply to Axiom forms. The refresh variables are only presented to the user when the user clicks the filter button. There is no way to force the user to select values when the form is first opened.

Axiom forms also support an additional refresh variable option, which is the ability to group variables for display in the Filters panel. For more information, see the following section on *Grouping refresh variables*.

In order for the filter icon to be present on the Web Client Task Bar, a RefreshVariables data source must be present in the file and must contain at least one enabled variable. The RefreshVariables data source does not have to be linked to anything on the form canvas.

Grouping refresh variables

When using refresh variables with Axiom forms, you can optionally use the `[GroupName]` column in the RefreshVariables data source to group variables within the Filters panel. Grouped variables display within an expandable / collapsible section, using the group name as the section header.

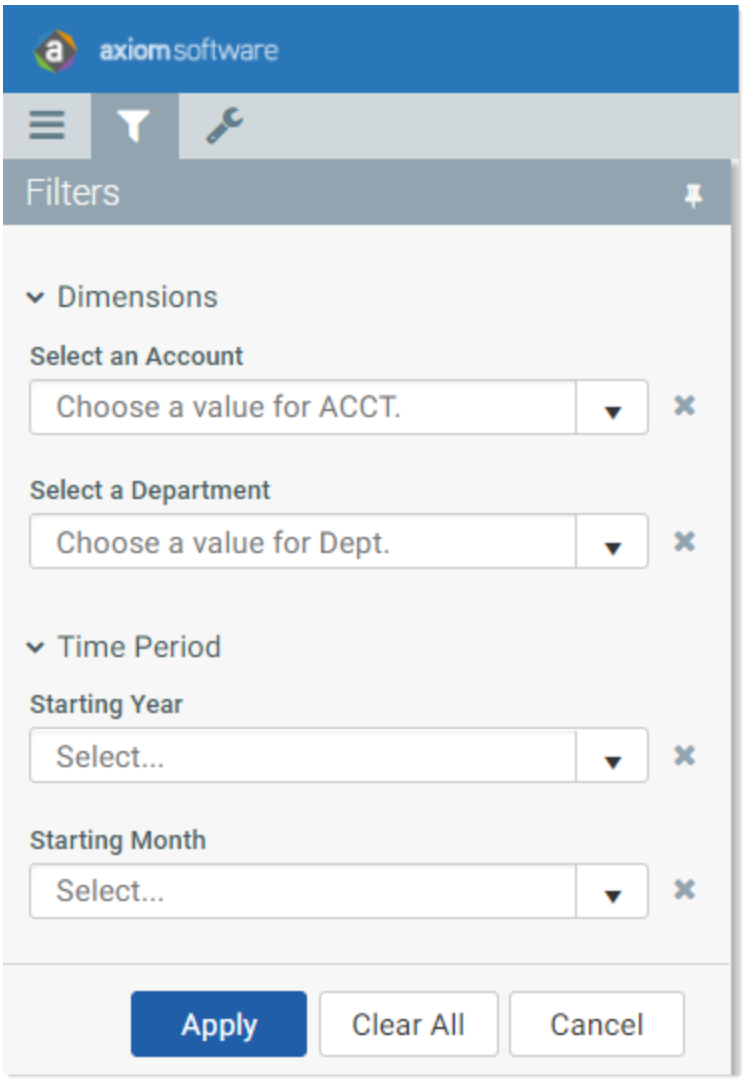
Groups display in the order they are found in the RefreshVariables data source. All variables belonging to that group name will display in that group, in the order the variables are listed in the data source. Therefore, when using groups, a variable that is lower in the list may display before variables above it, depending on its group assignment.

When using groups, it is recommended to assign all variables to a group. Any variables without a group assignment will display after the grouped variables.

The following example data source shows several variables belonging to groups:

	A	B	C	D	E	F	G	H	I
4									
5			[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsEnabled]	[SelectedValue]	[GroupName]
6			[Variable]	Acct	Select an Account	ComboBox	TRUE		Dimensions
7			[Variable]	Dept	Select a Department	ComboBox	TRUE		Dimensions
8			[Variable]	Year	Starting Year	StringList	TRUE		Time Period
9			[Variable]	Month	Starting Month	StringList	TRUE		Time Period

When a user views this file as a form and opens the Filters panel, the variables display as follows:



You can also use the `[CollapseOnOpen]` column (True/False) to specify that a group starts out as collapsed. By default, groups are expanded, but you can use this option to collapse groups as needed. If one variable in a group is set to True, then the group starts out as collapsed.

Filters panel behavior

When a user clicks the filter button in the Web Client Task Bar, all enabled variables are rendered in the Filters panel. If no groups are used, variables are presented in the order they are defined in the data source. If using groups, see the previous section for information on how variables are ordered.

The variables start at the top of the Filters panel and display vertically downward. If the number of variables exceeds the height of the panel, a vertical scroll bar is present.

Once the user has made selections for all required variables and any optional variables, the user can click the **Apply** button to submit the variable values to the form source file and trigger the data refresh. This process works as follows:

- The **Triggering Component** for the form is set to **Axiom.RefreshPanel**. This is a reserved name that identifies the Filters panel as the source of the refresh. You can use this information to dynamically enable Axiom queries to execute as part of the refresh.
- The user's selected values for the refresh variables are placed in the `[SelectedValue]` column of the data source.
- Axiom queries that are active and set to **Refresh on Manual Refresh** are run. The file is calculated before and after the query refresh. Data lookups will be run as normal (unnamed data sources are always executed after the Axiom queries as part of the manual refresh, and named data sources are executed after the Axiom queries if configured to do so).

  Just like when using refresh variables in spreadsheet Axiom files, it is up to the form designer to set up the data queries so that they change based on the user's selected values. For example, the data filter of an Axiom query may change depending on the user's selected value for a variable.
- If the form contains a Data Grid component, the data query for the grid is refreshed using the current state of all component and data source settings. Again, it is up to the form designer to set up the data grid properties to change based on the user's selected values. Any user changes made to the grid in the form—such as filtering columns, sorting columns, selecting rows—are all reset due to the refreshed data.
- The form is refreshed in the browser to reflect the current state of all components.

IMPORTANT: The Filters panel is intended to impact data queries only. The Filters panel does not trigger the full form update cycle. For example, interactive values for components are not submitted back to the source file, and save-to-database cannot be triggered. Any component changes that have not yet been submitted to the source file will be lost after the refresh, and the component will be reverted to its current state in the source file.

The variables and their current settings are read from the source file when the form is first opened, and after that, every time the variable values are submitted using **Apply**.

Dependent variables behave the same way that they do in the Desktop Client. If a user changes the value of a variable that another variable is dependent on, the selected value of the parent variable is sent to the source file and the file is calculated. Any changes to the dependent variable will then be reflected in the Filters panel, including showing or hiding variables based on the `[IsEnabled]` column as appropriate. This process only updates the refresh variables in the panel; the form itself is not updated.

If the RefreshVariables data source changes due to an update triggered by an interactive component in the form, the Filters panel will *not* update for these changes until the variable values are submitted using Apply. For example, imagine that you have a variable that uses a formula to determine whether the variable is enabled or not, and that formula looks to the selected value of a ComboBox component. If the user changes the value of the ComboBox component and this change causes the variable to become

enabled, the Filters panel will not immediately update for this change. However, when the user clicks Apply to submit the refresh variables, the panel will be updated after the data refresh, and at that point the new variable would show up. Therefore, it is not currently possible to adjust the RefreshVariables data source using form components, as the change will not be recognized in the Filters panel as part of the form update triggered by the component.

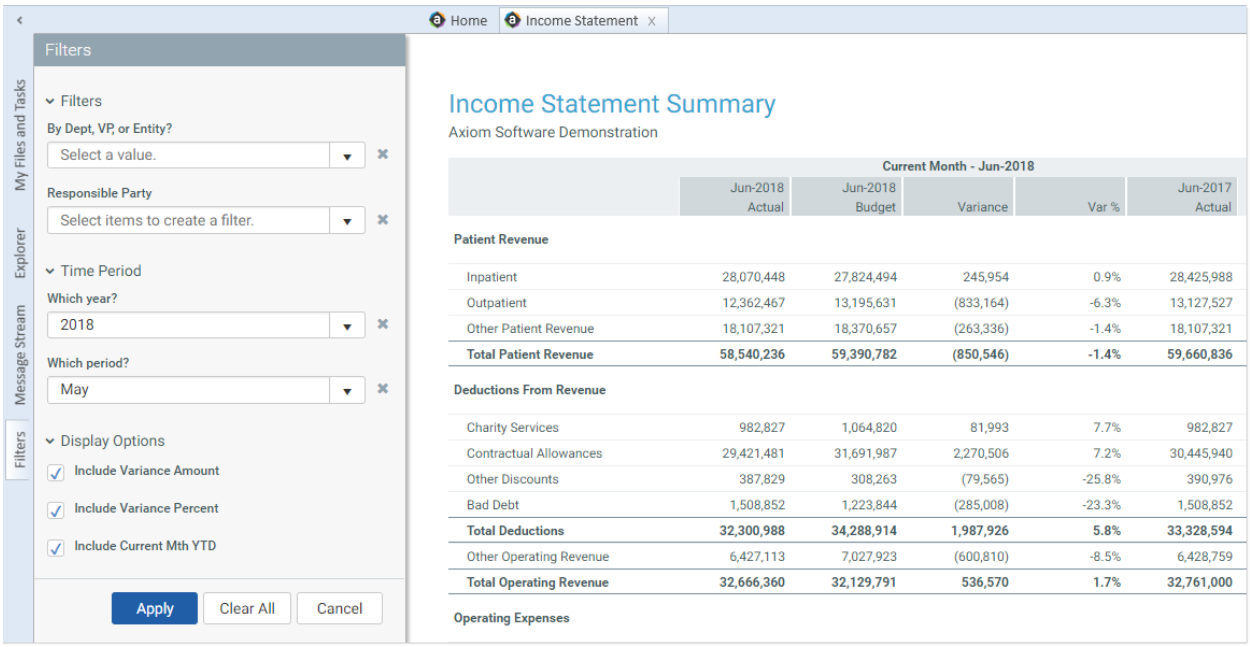
Filters panel behavior for composite forms

If you are using a composite form with one or more Embedded Form components, the Filters panel shows refresh variables for the parent form and the currently visible child embedded forms. When the variables are applied, they are applied to their respective parent or child forms and the forms are refreshed as described in the previous section. Note the following:

- Variables from the parent form are displayed first, followed by variables from the currently visible child forms. It may be helpful to use groups to organize the variables if there are many variables or if you need to distinguish between the parent and child variables. However, keep in mind that variables cannot be grouped across forms. For example, if the parent form has a group called Time Period and the child form has a group called Time Period, the variables will not display in the same group. Instead, the Filters panel will display duplicate group names.
- Variables in one form cannot depend on variables in other forms. For example, you cannot use the selected value of a variable in the parent form to filter the list for a variable in the child form. It is possible to use shared variables to pass the selected value of one form's variable to another form, but the value cannot be used to impact a refresh variable in the other form.
- When refresh variables are applied, the parent form is refreshed first, followed by the child form (assuming both forms have refresh variables). If only one form has refresh variables, then only that form will be refreshed by the Filters panel. The other form will not be updated unless you force it to do so by using an option such as **Refresh Parent Form** or **Force Refresh**.
- As noted in the previous section, any unsubmitted changes in the parent or child form will be lost when refresh variables are applied to that form. However, if the parent or child form does not have refresh variables and is not otherwise being forced to update, then the form will be left as is, including its unsubmitted changes. If the parent or child form does not have refresh variables but is being forced to update by the other form, then changes will be submitted as normal during that update.

Using the Filters panel in the Desktop Client

The Web Client Container is not available when an Axiom form is opened as a web tab within the Desktop Client. If the form has defined refresh variables, then the Web Client Filters panel automatically opens and displays along the left side of the form, as if it were a task pane in the Desktop Client.



Example Filters panel in the Desktop Client

Form users in the Desktop Client can use the Filters panel to select variable values and refresh data in the form, just as they can in the Web Client. Users can switch between other task panes and the Filters panel by clicking the side tabs. To hide the Filters panel, users can collapse the entire Axiom Assistant area by clicking the button in the header.

This behavior is only available if the form meets the following conditions:

- The Web Client Container is enabled for the form, and refresh variables are defined in the form. Even though the container does not display in this environment, enabling the container means that you want the form to have access to the container features such as the Filters panel.
- The form is open as a web tab within the Desktop Client. This behavior is only available in the Windows Client. If you are using the Excel Client, then Axiom forms open within the browser instead of within the Desktop Client application. Forms opened in the browser have access to the Web Client container as normal.

  NOTE: In the Excel Client, only a form-enabled Home file will open as a web tab within the Desktop Client application. In this case, the Filters panel will be available in the Desktop Client for the Home file, if applicable.

If a form is opened in the Desktop Client as a dialog or a task pane, then refresh variables are ignored and the Filters task pane will not be present.

Displaying announcements in Axiom forms

You can use the Announcement component to display announcements to users in Axiom forms. This is typically included as part of a form-enabled Home file, to be used as either the Desktop Client Home file, or as the Web Client Forms Home page.

The Announcement component is a self-contained solution for announcements. Using the component, you can display, create, edit, and delete announcements.

Enabling announcements

To enable use of announcements, you must have at least one form-enabled file with an Announcements component. When the form is rendered to users, this component enables both the display and management of announcements. You might have one Home file with one Announcements component, or several different files targeting different audiences.

All Announcements components in your system use the same announcements repository stored in the Axiom Software database. If you have one Announcements component in one file, and another in a different file, both components have access to the same announcements for display. If you add a new announcement using one component, that announcement is available to all components in the system.

Although every Announcement component has access to the same set of announcements, you can configure a particular component so that it only displays certain categories of announcements. This enables the ability to show different announcements to different sets of users. For example, you might have announcement categories of Budget and Capital, and configure product-specific Home files with Announcement components that only show the announcements for the applicable category. You could do the same thing with announcement categories of Finance and Dept Manager, to display announcements for role-specific Home files.

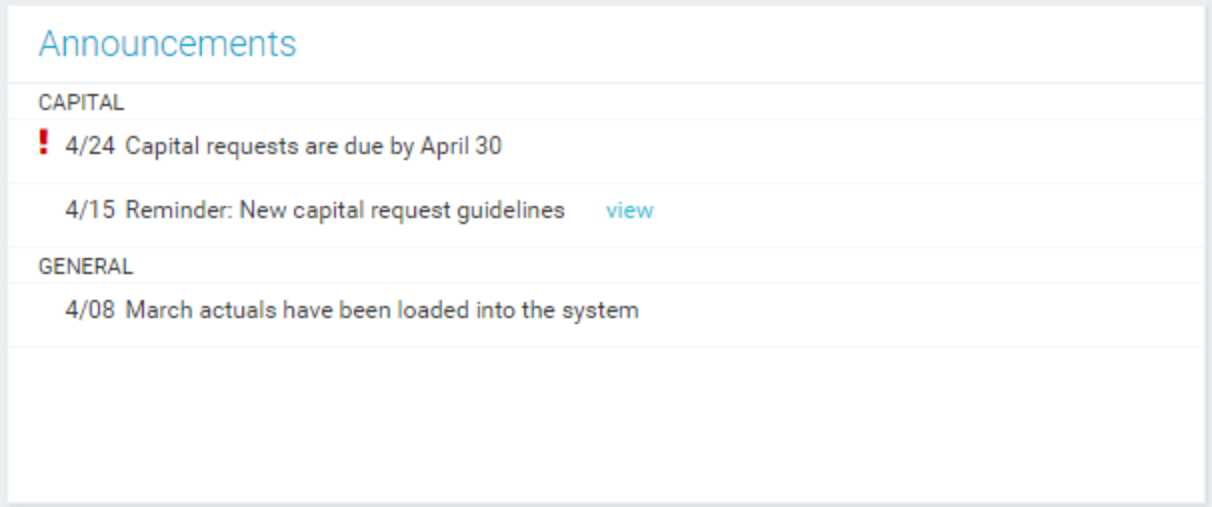
When you place an Announcement component on a form, you configure whether it displays all announcements, or only certain categories of announcements. For more information on configuring the Announcement component, see Announcements component .

Viewing announcements

For end users to view announcements, they need access to a form-enabled file that contains the Announcements component. This could be their assigned Home file, or it could be some other Axiom form that is designed to communicate announcements to end users.

All users can see all announcements that the Announcement component is configured to display. There are no security permissions required to view announcements (other than access to a file with the Announcements component).

Announcements display as shown in the following screenshot. Announcements are sorted by date (within each category if categories are used), with the newest announcements displayed on top of the list. If the announcement has message text or if the title is too long to display in the row, the user can click **view** to see the full text of the announcement.

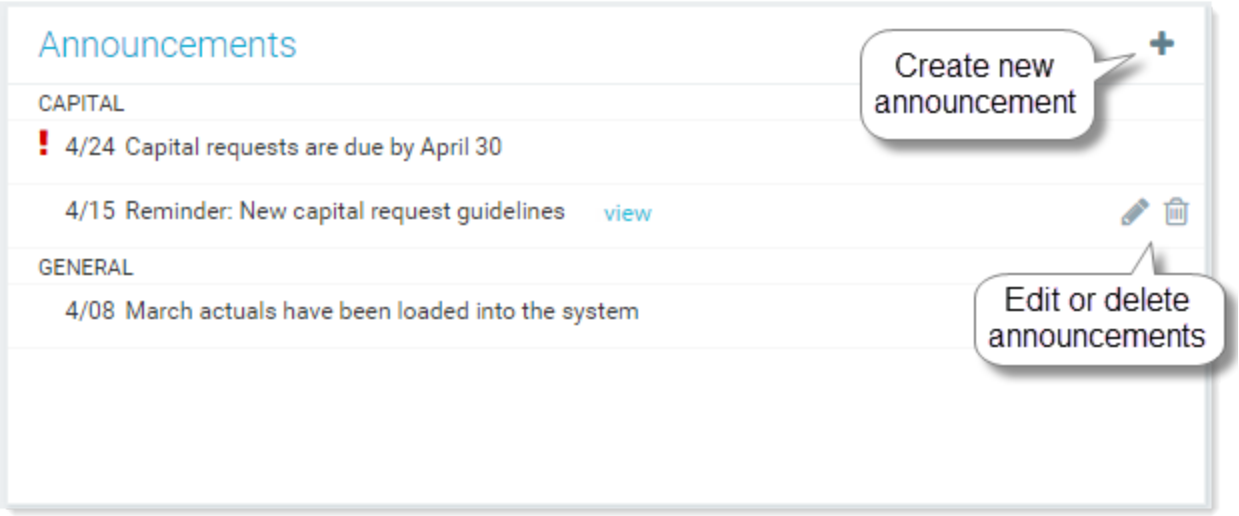


End users cannot perform any actions on announcements. End users cannot delete announcements or mark them as read. Announcements will continue to display in the component until they are deleted by an announcement administrator, or until the announcement's expiration date is reached.

Managing announcements

Administrators and users with the Administer Announcements security permission can add, edit, and delete announcements using the Announcement component. These users need access to a form-enabled file that contains the Announcements component. Although announcement administrators may already have access to a home page with an Announcements component, it is recommended to provide a separate "management" form specifically for the purpose of managing all announcements.

When a user has permission to administer announcements, the Announcement component provides additional controls. The user can click the plus icon in the top right corner to create new announcements, and they can edit or delete any announcement displayed in the component.



Managing announcements

Using the Announcement component in an Axiom form, you can add, edit, and delete announcements.

In order to provide full announcement management capabilities, it is recommended to create an Axiom form that is only intended for use by announcement administrators. This form should contain an Announcement component that is configured as follows:

- Enable **Show All Announcements**, so that announcement administrators can view and edit future and expired announcements. Otherwise, future and expired announcements will not display in the component and cannot be accessed.
- Size the component to a tall height so that announcement administrators can view many announcements without scrolling.

NOTE: The ability to manage announcements is only available to administrators and to users with the **Administer Announcements** security permission. Users without these permissions only see announcements within the component; they do not see any of the editing controls.

Adding announcements

You can add announcements using any Announcement component in an Axiom form. All announcements created using a component are saved to the same central repository and are available to all Announcement components in the system. It does not matter whether the current Announcement component is filtered to only showing certain categories; you can still use the component to create a new announcement for any category.

To add an announcement:

1. Click the plus icon in the top right corner of the Announcements component.
2. In the **Add Announcement** dialog, complete the following announcement properties:

Item	Description
Category	The category for the announcement. All announcements must have a category. Categories can be used to display different announcements to different audiences, by configuring Announcement components to only show announcements for certain categories. By default, the first category in the list is selected. If you want to use a different category, select it from the list. If you need to create or edit a category, click **Manage Categories**. If your organization is not using different categories, then all announcements are assigned to the default **General** category.

Item	Description
Start Date	The date that you want the announcement to start displaying to users. By default, this is today's date. If you do not want the announcement to start displaying until some point in the future, then you can select a future date. The announcement will not display in any Announcement components until that date is reached (unless **Show All Announcements** is enabled for the component).
Expire Date	Optional. The date that you want the announcement to stop displaying to users. You can specify a date so that the announcement is automatically hidden once the expiration date is reached, or you can leave this blank so that the announcement continues to display until it is deleted by an announcement administrator. The expiration date must be after the start date, and cannot be today's date.
Title	The title of the announcement. The title text is what displays to users in the announcement component. The title text may be the entirety of the announcement. For example, you could define a title such as "Reminder: All budgets due today!" with no additional message text. If the title text is too long to display in the Announcement component (this depends on the current width of the component in the form), then an ellipses displays at the end of the visible text. Users can click the **view** link to see the full announcement text.
High Importance	Optional. Select this check box if you want the announcement to display with high importance. In the Announcement component, a red exclamation point displays next to the announcement.
Message	Optional. The message text of the announcement. Use the text box to define the message text. The text can be multiple lines and can use bold, italic, and underlined text. Message text does not display within the Announcements component. If message text is defined, the announcement has a **view** link. Users can click this link to view the full announcement text (title and message).

3. Review all announcement settings to ensure they are as you want them, and then click **Save**.

If the start date is today, the saved announcement is immediately available to all Announcement components, and will be visible in any components that are configured to display the announcement's assigned category. If a user is currently viewing a form with an Announcement component, the new

announcement will not display until the form is updated by using a Button component or any other component configured to auto-submit.

Editing announcements

You can edit any announcement that is currently displayed in an Announcement component in an Axiom form. It is recommended to use an Announcement component that is configured with **Show All Announcements** enabled, so that you have access to all available announcements.

If the Announcement component that you are using for editing does not have Show All Announcements enabled, then you will not be able to edit the following announcements because they do not display in the component:

- Announcements with start dates in the future
- Announcements that have reached their expiration dates
- Announcements that have been filtered from display by using the **Limit Categories To** option.

To edit an announcement:

1. Hover your cursor over the row of the announcement that you want to edit, so that the editing icons display on the right side of the row. Click the pencil icon to edit the announcement.
2. In the **Edit Announcement** dialog, edit the announcement settings as desired. See the previous section for more information on the announcement settings. Note the following:
    - Changing the announcement category may cause the announcement to no longer display in certain components, and to start displaying in other components, depending on which categories those components are configured to show.
    - Changing the start date to the future will cause the announcement to stop displaying in any components (except components with **Show All Announcements** enabled).
    - It is not possible to change the expiration date to today to hide the announcement immediately. You must delete the announcement if you want it to stop displaying today.
3. Review all announcement settings to ensure they are as you want them, and then click **Save**.

Changes to announcements take effect immediately. If a user is currently viewing a form with an Announcement component, the changes will not display until the form is updated by using a Button component or any other component configured to auto-submit.

Deleting announcements

You can delete any announcement that is currently displayed in an Announcement component in an Axiom form. It is recommended to use an Announcement component that is configured with **Show All Announcements** enabled, so that you have access to all available announcements.

If the Announcement component that you are using for deletion does not have Show All Announcements enabled, then you will not be able to delete the following announcements because they do not display in the component:

- Announcements with start dates in the future
- Announcements that have reached their expiration dates
- Announcements that have been filtered from display by using the **Limit Categories To** option.

If you delete an announcement, it is not recoverable. Be sure that you no longer need the announcement before you delete it.

NOTE: When an announcement expires, it is not deleted from the Axiom Software database. Expiration simply means the announcement no longer displays in the component (except for components with Show All Announcements enabled). In order to remove the announcement from the database, you must delete it.

To delete an announcement:

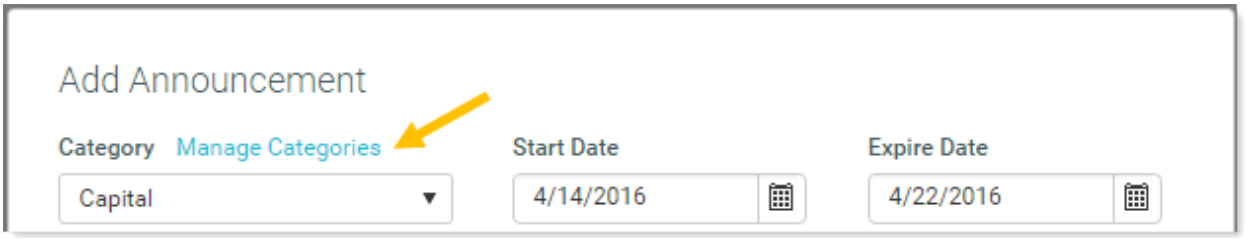
- Hover your cursor over the row of the announcement that you want to delete, so that the editing icons display on the right side of the row. Click the trashcan icon to delete the announcement.

The announcement is deleted immediately. If a user is currently viewing a form with an Announcement component, the announcement will not be removed until the form is updated by using a Button component or any other component configured to auto-submit.

Managing announcement categories

All announcements must belong to a category. If you are not using multiple categories, then all announcements are assigned to the default category of General. If you want to use other categories, you must create them.

Categories can be created, edited, and deleted from the **Add Announcement** or **Edit Announcement** dialog. Click the **Manage Categories** link above the **Category** drop-down list.



In the **Manage Categories** dialog, all existing categories display in a grid at the top of the dialog. Within this dialog, you can do the following:

- To add a new category, click **Add a New Category**. Define a **Name** and **Display Text** for the category, then click **Save**. The new category is added to the grid. The category name must be unique.
- To edit a category, click the category name in the grid, so that the name and display text display below. Edit the display text as needed and then click **Save**. The category name cannot be changed.
- To delete a category, hover your cursor over the category row until the editing icons display in the right side of the row. Click the trash can icon to delete the category.

  You cannot delete a category if active announcements are assigned to that category—you must delete these announcements or assign them to another category first.

The name of the category is used to identify the category, whereas the display text displays to users. You could define a category name of Cap and then display text of Capital. You can later edit the display text as needed, such as to Capital Planning, without changing the underlying ID of the category.

NOTE: When configuring an Announcement component to filter by category, the list of categories is loaded when the file is opened. If you add or remove a category, you must close and reopen the file in order to see the changes.

Executing Scheduler jobs from an Axiom form

You can execute a Scheduler job from within an Axiom form by using the RunEvent command.

Requirements and limitations

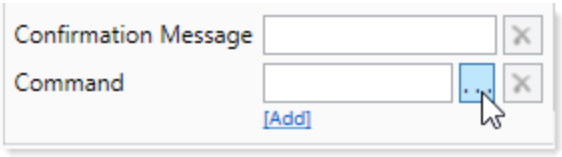
The setup for the Scheduler job that you want to run is the same as when using the RunEvent function in Axiom files. You must define an event handler within the job, and then configure the job as desired, including any variables that you want to pass in when the job is executed.

Setting up RunEvent within an Axiom form

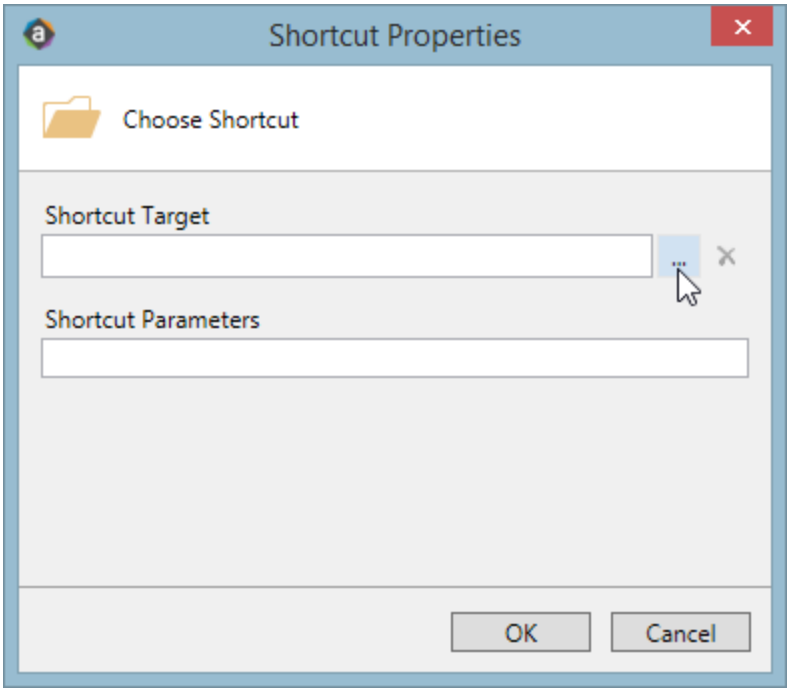
To allow users to execute a Scheduler job from the Axiom form, you use a Button component that is configured to run the RunEvent command shortcut. This is the same command that is available for use in task panes.

To start off, add the component to the Axiom form canvas and then configure the properties as desired. You will probably want the text for the component to be something like "Process Plan Files" or "Process Monthly Reports" (depending on what the target Scheduler job is doing). You can then configure the **Command** for the component as follows:

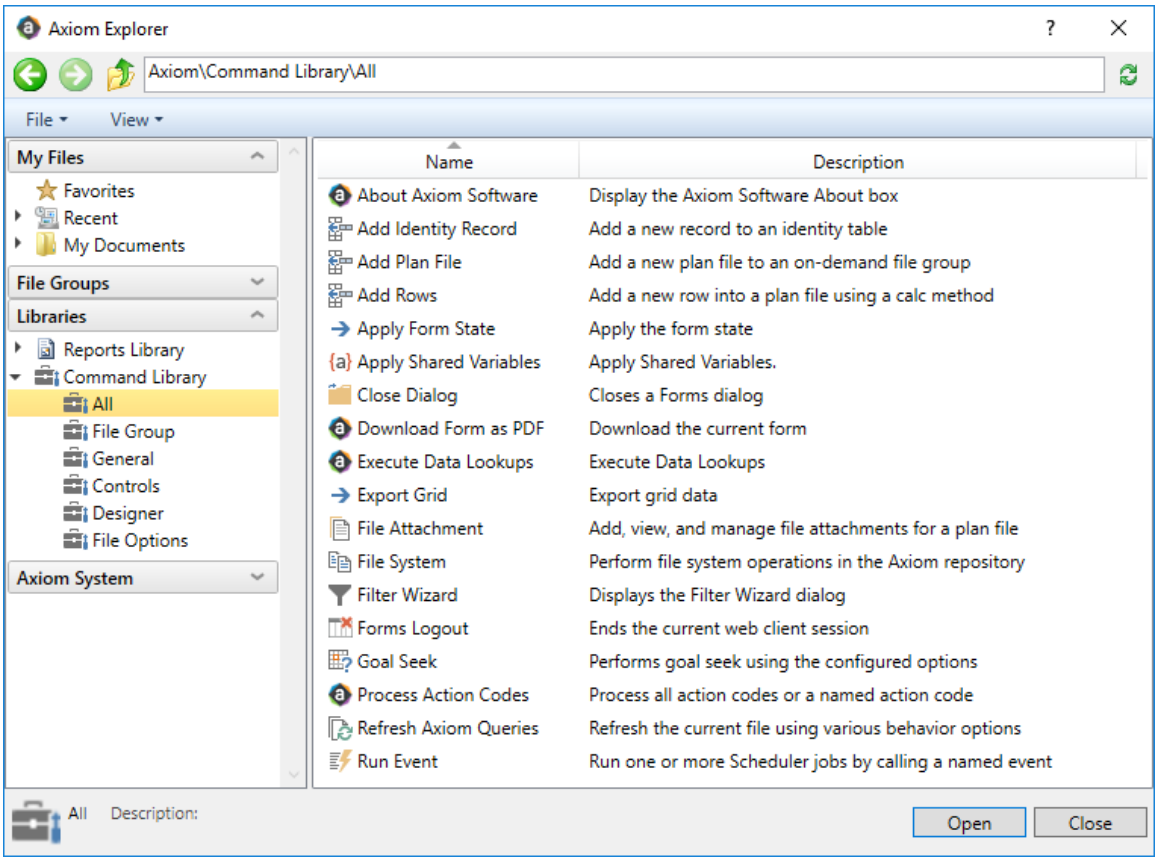
1. In the component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



3. In the **Axiom Explorer** dialog, select the **Command Library**, then select **RunEvent**, then click **Open**.



In the Shortcut Properties dialog, the RunEvent command is now listed as the shortcut target, and the relevant shortcut parameters are now available.

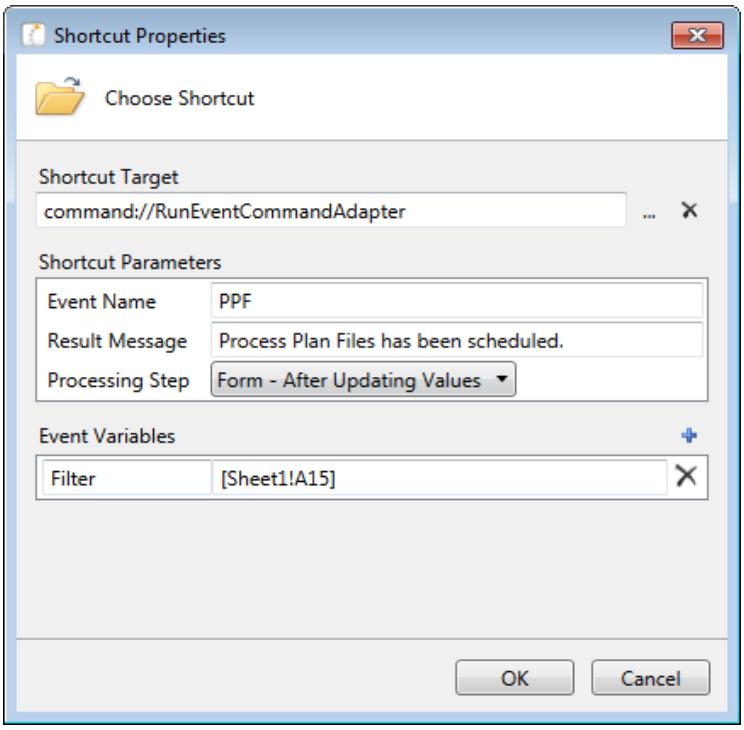
4. Complete the shortcut parameters as follows:

Parameter	Description
Event Name	Required. The name of the event handler that you want to trigger. Any job with an active event handler of the same name will be scheduled for execution.
Result Message	Optional. A confirmation message to display to the user after the jobs have been scheduled. If omitted, the standard generic message is used.

Parameter	Description
Processing Step	Optional. Specify the desired **Processing Step** for the command. By default, this is set to **Form - After Updating Values**, which means that the command will be performed after any changed values in the Axiom form have been submitted back to the source file. If desired, you can specify a different point in the refresh process for the command. For more information on when each process step occurs during the form refresh process, see Axiom form update process. **NOTE:** If you are using Event Variables, then the processing step for the command must be set to After Updating Values or later in the process. If the processing step is set to Before Processing, then the command will execute before any changed values are submitted back to the source file.

NOTE: In other contexts, the shortcut parameters for the RunEvent command allows for defining a confirmation message. This parameter does not display for Axiom forms, because the confirmation message should be defined on the Button component properties instead.

5. Optional. If you want to pass variable values to the Scheduler job, you can do this using the **Event Variables** section.
    - Enter the variable name in the left-hand box. This variable must be defined in the target Scheduler job and used in the job settings.
    - Enter the variable value in the right-hand box. You can hard-code a value, or you can enter a cell reference in brackets. For more information, see the following section.
    - If you need more rows to define additional variable values, click the plus icon ✚.



Example Shortcut Properties dialog

6. Once you have finished configuring the Shortcut Properties, click **OK** to close the dialog and return to the component properties.

Within the Axiom form, the user can click the button to perform the RunEvent command. Axiom Software will look for any jobs with a matching event handler name and place them on the schedule to be eligible for immediate execution (pending available Scheduler threads and any higher-priority jobs already in the queue). The result message will display in the bottom left-hand corner of the form.

Collecting variable values from the Axiom form

If desired, you can set up the Axiom form to collect variable values from the user, and then pass those values to the Scheduler job via the Event Variables. To do this, you must:

- Place one or more interactive components in the Axiom form to collect the input from the user. For example, you might use a Text Box or Combo Box component, or you might use an editable cell within a Formatted Grid component.

- When configuring the shortcut parameters for the RunEvent command, you should designate a cell reference in brackets as the variable value. Axiom Software will read the variable value from the designated cell.

  If the component you are using to collect user input is a formatted grid, then the cell reference can simply point to the editable cell in the formatted grid. If the editable cell is A15 on Sheet1, then you enter [Sheet1!A15] for the variable value.

  If you are using a different interactive component, such as a text box, then you should use an indirect reference for the updateable property of the component, so that the value is written to and read from a cell in another sheet. For example, for the Text property of the text box, you can enter [Sheet1!A15]. This means that the text box value will be written to and read from that cell, instead of the Text cell on the Form Control Sheet. You would then also enter [Sheet1!A15] as the variable value.

  You should use this indirect behavior instead of referencing a cell on the Form Control Sheet directly, because any time a new component is added to or deleted from the form, that cell reference may change.

The Axiom form user can complete these inputs before pressing the button to trigger the RunEvent command. There is no way to require the user to complete these inputs, however, you can use the Confirmation Message property of the button to remind the user to complete the inputs and confirm that they want to continue.

When the user presses the button in the form, the following occurs:

- If a Confirmation Message is defined for the button, that message is presented and the user must click OK to continue (or Cancel to cancel, in which case none of the following steps occur).
- The changed values in the form are submitted back to the source file.
- The RunEvent command is executed (this assumes the Processing Step for the command is set to After Updating Values).
- The refresh process for the form then continues as normal.

Processing action codes in an Axiom form

You can process action codes in an Axiom form by using the Process Action Codes command.

Action codes are always processed as normal in the form source file when Axiom queries are run or when a calc method is inserted. This command provides a way to process action codes "on demand" regardless of whether Axiom queries or calc methods are used in the sheet. This command is also the only way to process named ActionCodes tags.

NOTE: This topic discusses action code processing in a standard Axiom form, to process action codes on the form source file for the purpose of impacting the form displayed to the user. If you are using an Axiom form as a dialog in the Excel Client or Windows Client, then you can optionally use this command to process action codes on the active client spreadsheet instead of within the form source file. For more information, see Custom Dialogs and Task Panes in the Desktop Client and Executing commands on the active client spreadsheet from an Axiom form.

Requirements

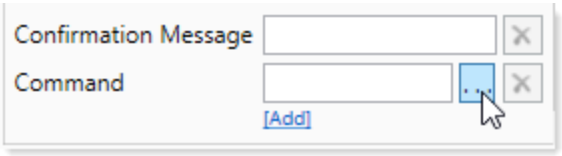
Action codes must be set up in the form source file, in the specified sheet. If no action codes are present, then an error will be presented to the Axiom forms user when they trigger the command.

Setting up action code processing for an Axiom form

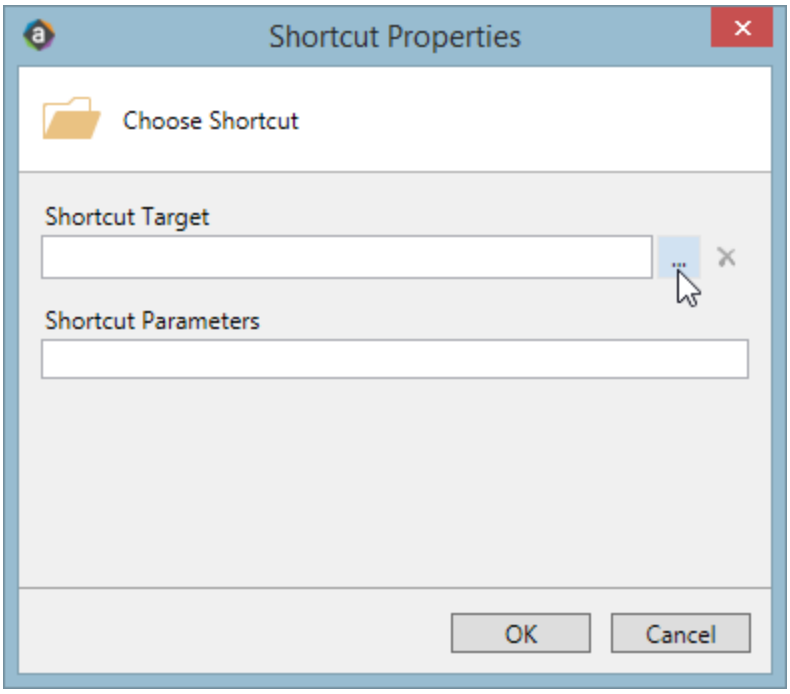
To allow users to process action codes from the Axiom form, you use a Button component that is configured to run the Process Action Codes command shortcut. This is the same command that is available for use in task panes.

To start off, add the component to the Axiom form canvas and then configure the properties as desired. You can then configure the **Command** for the component as follows:

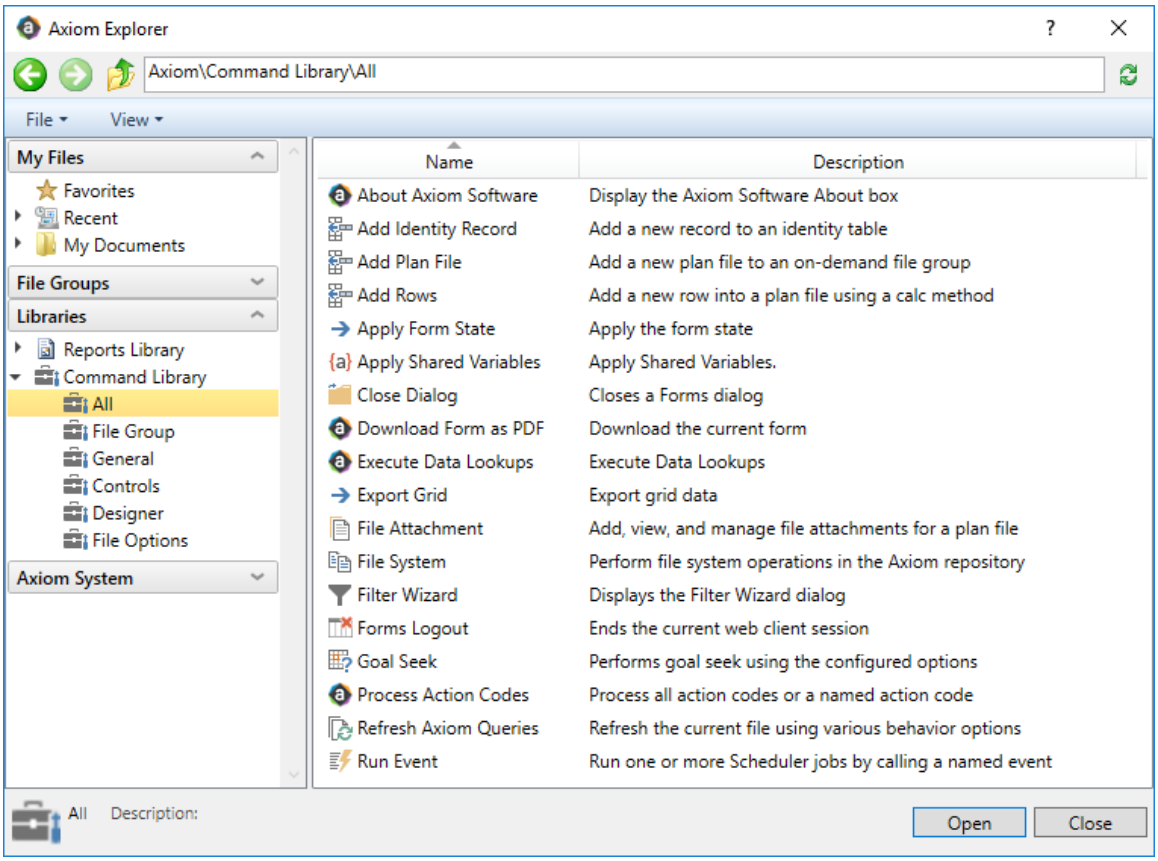
1. In the component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



3. In the **Axiom Explorer** dialog, select the **Command Library**, then select **Process Action Codes**, then click **Open**.

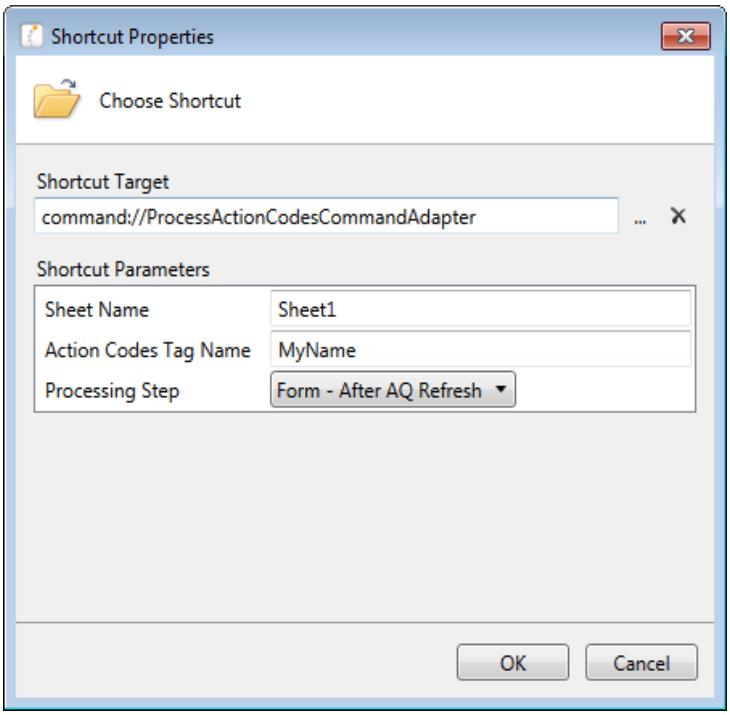


In the Shortcut Properties dialog, the Process Action Codes command is now listed as the shortcut target, and the relevant shortcut parameters are now available.

4. Complete the shortcut parameters as follows:

Parameter	Description
Sheet Name	Specify the sheet for which to process action codes. Make sure to specify a sheet name when using this command in Axiom forms. If this parameter is left blank, then action codes will be processed on the current sheet, which in this context means whichever sheet was active when the source file for the Axiom form was last saved.

Parameter	Description
Action Codes Tag Name	Optional. Specify the name of the ActionCodes tag that you want to process. To use this option, the sheet must contain an ActionCodes tag with a matching name. For example, if you enter `MyName`, then the sheet must contain a tag as follows: `[ActionCodes:MyName]`. The processing behavior is as follows: • If a tag name is specified, then *only* the action codes defined in that tag's control row and control column will be processed. If the sheet contains other ActionCodes tags (named or not), they will be ignored. • If no tag name is specified, then all action codes will be processed *except* those belonging to named ActionCodes tags.
Processing Step	Optional. Specify the desired **Processing Step** for the Process Action Codes command. By default, this is set to **After AQ Refresh**, which means that the Process Action Codes command will be performed after Axiom queries have been refreshed in the source file. If desired, you can specify a different point in the refresh process for the Process Action Codes command. For more information on when each process step occurs during the form refresh process, see Axiom form update process.



Example Shortcut Properties dialog

5. Once you have finished configuring the Shortcut Properties, click **OK** to close the dialog and return to the component properties.

Within the Axiom form, the user can click the button to perform the Process Action Codes command. Action codes will be processed at the specified processing step, and then the Axiom form refresh process will continue as normal. Remember that unnamed action codes are always processed when Axiom queries are run, so if any Axiom queries are run as part of the normal form refresh, those action codes will be processed at that point.

Action code behavior notes

When the Process Action Codes command is performed in the Axiom form environment, a calculation is performed before action codes are processed and then after action codes are processed. This means that you can set up your action codes with dynamic formulas, and those formulas will always calculate before action codes are processed, regardless of the selected Processing Step for the command.

Using the Filter Wizard in an Axiom form

You can make the Filter Wizard available in an Axiom form, so that form users can open the wizard to create a filter criteria statement. The filter criteria statement gets submitted back to a target cell in the form source file, where it can then be used to impact the data shown in the form. For example, you might reference the filter in the **Data Filter** setting of a particular Axiom query, or as a **Sheet Filter** to impact all queries on a particular sheet.

To do this, use the Filter Wizard command on a Button component (or on a Button tag in a thematic Formatted Grid component). Form users can click on the button to open the Filter Wizard and build the filter criteria statement. When configuring the command, you can specify the user interface for creating the filter—either Advanced mode or Hierarchy mode—and you can optionally limit the wizard to only showing certain tables or certain hierarchies.

You can also use the Filter Wizard command to allow users to create a limit query statement for an Axiom query, but this is an advanced feature only for use in certain special situations.

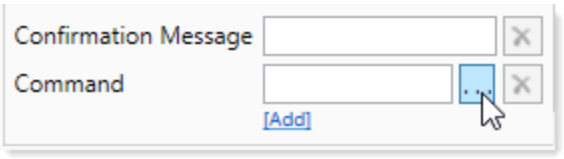
NOTE: The Filter Wizard command requires the Web Client Container to be enabled for the form. If it is not enabled, an error will occur when the command is executed. The Web Client Container is enabled by default for new forms.

Setting up a button to open the Filter Wizard

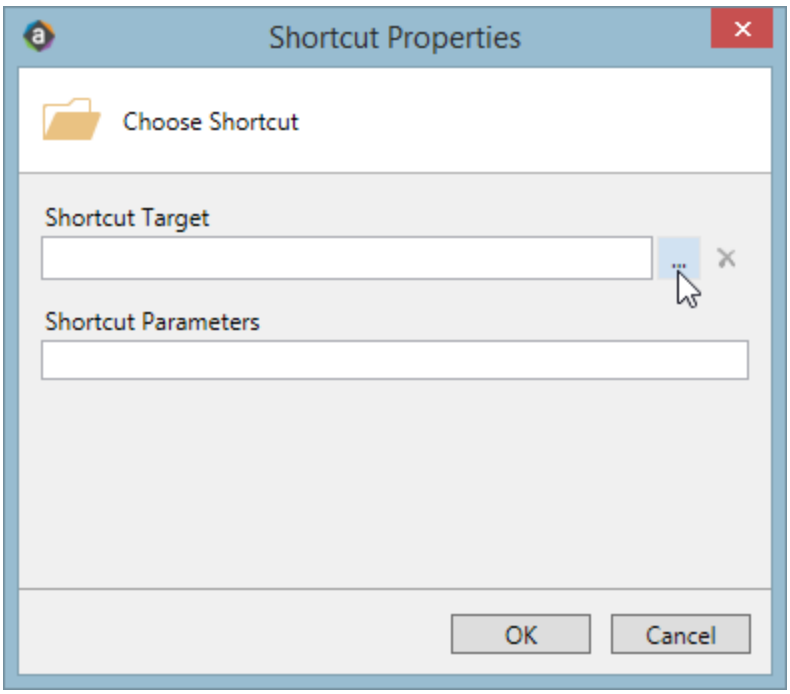
To allow form users to open the Filter Wizard and create a filter, use a Button component that is configured to run the Filter Wizard command.

To start off, add the Button component to the Axiom form canvas and then configure the button properties as desired. The button text should be set to something like "Define Filter" or "Filter Data". You can then configure the **Command** for the button as follows:

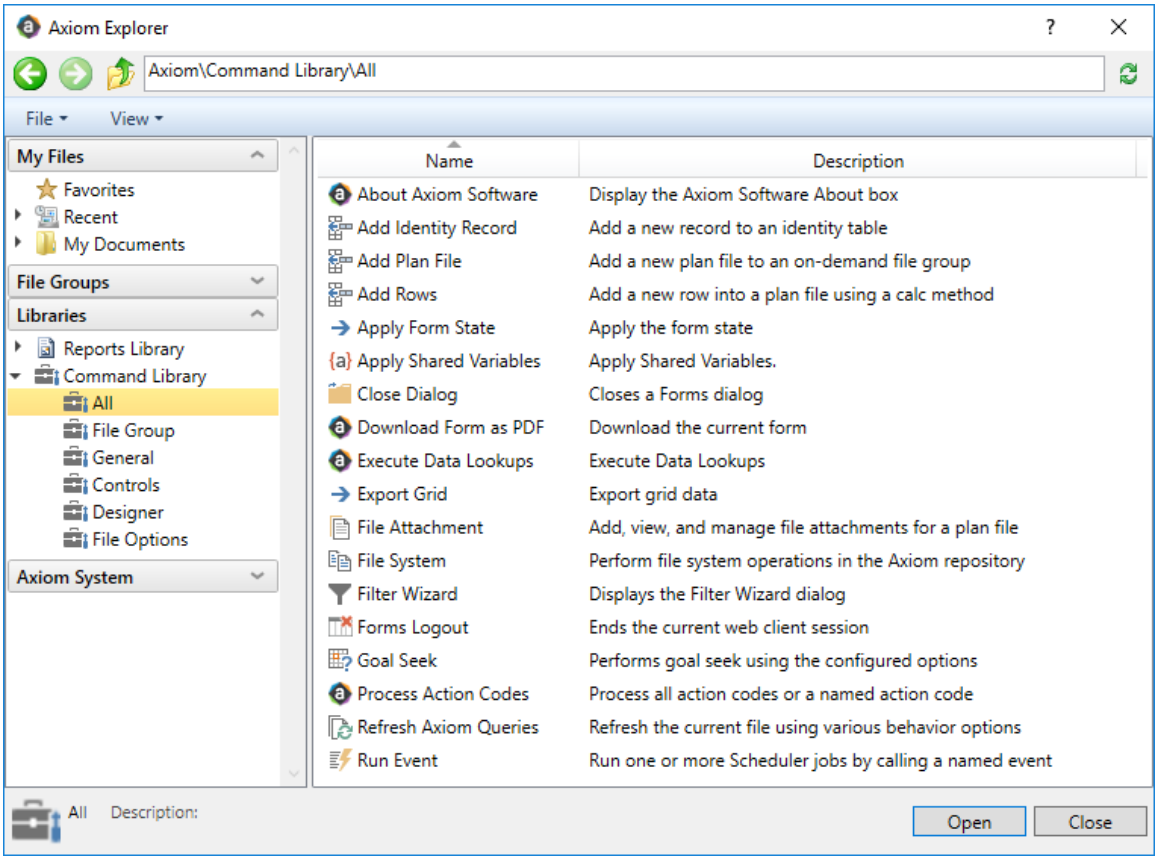
1. In the component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



3. In the **Axiom Explorer** dialog, expand the **Command Library** and then locate the **Filter Wizard** command in the library. Select the command and then click **Open**.



In the Shortcut Properties dialog, the Filter Wizard command is now listed as the shortcut target, and the relevant shortcut parameters are now available.

4. Complete the shortcut parameters for the command as follows, and then click **OK** to close the Shortcut Properties dialog.

Item	Description
Target Cell	The target cell in the source file to place the filter after it has been created (such as `Sheet1!A1`).
Type	Select one of the following to determine how users can create the filter: • **Advanced Filter** (default): Users can create the filter based on literal table column names and values. The Advanced View option also supports creating limit query statements for use in Axiom queries. • **Hierarchy Filter**: Users can create the filter based on hierarchy levels in defined hierarchies.

Item	Description
Tables / Hierarchies	Optional. Specify one or more tables or hierarchies to limit the options shown in the Filter Wizard. Valid entries depend on the specified filter type: • If using **Advanced Filter**, then you can enter one or more table names, separated by commas. The Filter Wizard will only show columns from the specified tables, as well as any lookup tables. For example, if you list GL2018, and that table has lookups to Acct and Dept, then all three tables will be available in the wizard. • If using **Hierarchy Filter**, then several different options are available to specify the hierarchies to be shown. See the discussion following this table for more information. If left blank, then the Filter Wizard shows all available options for the specified filter type. For Advanced Filter this means all tables, and for Hierarchy Filter this means all hierarchies. **NOTE:** If desired, you can read the list of tables or hierarchies from a designated cell in the source file instead of entering the list into the shortcut parameters. To do this, use a cell reference enclosed in brackets, such as: `[Sheet1!A1]`. When using this approach, you can dynamically change the list by using a formula in the specified cell.
Limit Filter Column	Optional. In most cases, you will leave this property blank. Only complete this property if you want to use the Filter Wizard to create a limit query statement for an Axiom query.
Processing Step	Specifies when the command will be executed during the Axiom form update process. This option is set to **Form - After Updating Values** and cannot be changed. For more information, see Timing of command execution. **NOTE:** This command has special behavior and does not follow the normal form update process. For more information, see Form update behavior when using the Filter Wizard.

When using the **Hierarchy Filter** option, the following entries are valid in the **Tables / Hierarchies** parameter.

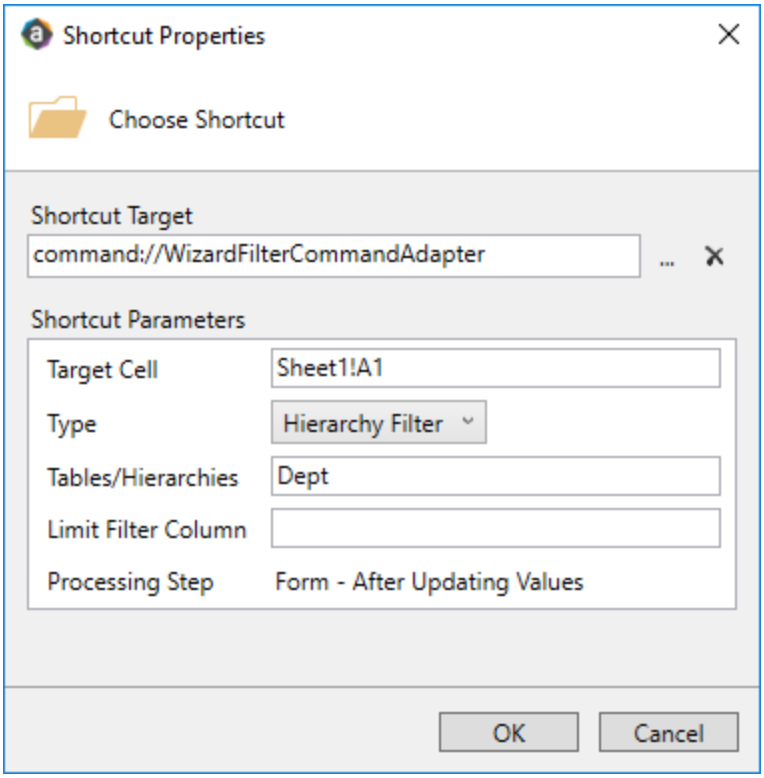
- Enter a table name to display all hierarchies defined for that table. For example, enter DEPT to display all hierarchies defined on the DEPT table.

  You can also enter multiple table names, separated by commas. The dialog will display all hierarchies defined for all listed tables.

- Enter *Table*:*HierarchyName* to only show the specified hierarchy. For example, DEPT:Geography to only show the Geography hierarchy on the DEPT table.

  You can also enter multiple table:hierarchy pairs, separated by commas. The dialog will display all specified hierarchies.

- Enter *Table*.*Column*:*HierarchyName* to only show the specified hierarchy and also use the specified Table.Column in the resulting filter criteria statement. For example, DEPT.Region:Region to show the Region hierarchy on the Region table, where DEPT.Region looks up to the Region table. The resulting filter criteria statement will be written such as Dept.Region.RegionType=1 instead of Region.RegionType=1, thereby allowing the filter to be applied to tables with a lookup to DEPT.



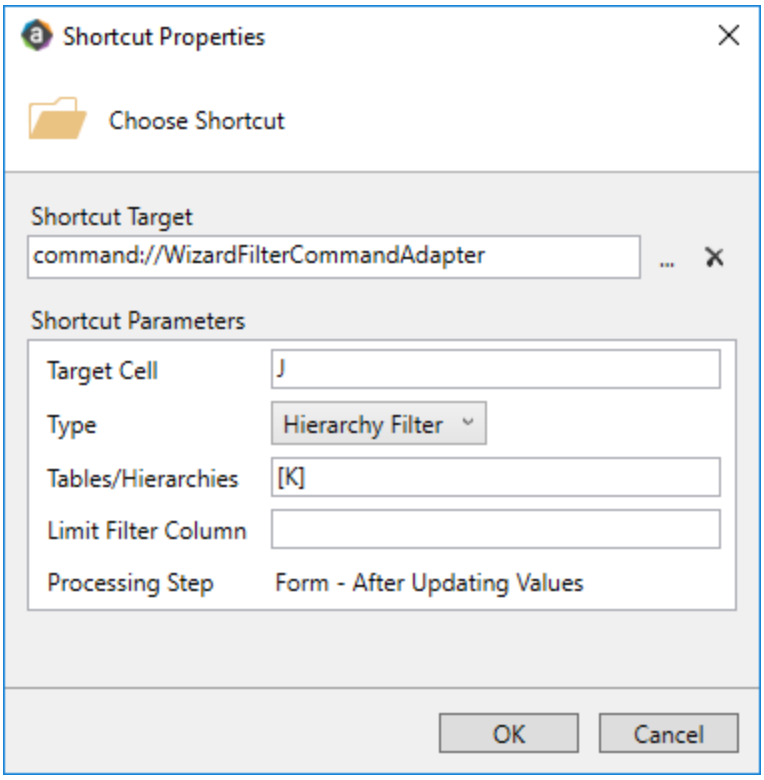
Example Shortcut Properties dialog

Using a Button tag in a Formatted Grid component

Button tags in thematic Formatted Grid components can also be configured to run this command. In this case, use the Command parameter within the tag to assign the command to the button. The easiest way to do this is to use the Tag Editor dialog or the Data Source Assistant to create the tag and edit the tag parameters. When using these helper dialogs, you can select the command and configure the shortcut parameters using the same method described previously for the Button component.

When using a Button tag, you can optionally specify the **Target Cell** using just a column letter, instead of a full cell reference. For example, you can specify J to indicate that the target cell should be in column J in the current row of the grid.

This column letter syntax can also be used for the **Tables / Hierarchies** parameter, if you want to read the list from the sheet instead of entering it in the shortcut parameters. In this case, enter the column letter in brackets, such as `[K]`.



Example column letter entries that are valid for use with Button tags

Form update behavior when using the Filter Wizard

When using the Filter Wizard command on a button, the full form update process does not apply. Instead, the following occurs when a user clicks the button:

1. The form update process proceeds as normal until it reaches the After Updating Values processing step, then it is aborted.

   Due to this abbreviated update process, the button cannot use **Save on Submit**, and cannot execute any other commands at later processing steps.

2. The Filter Wizard dialog opens and the user creates a filter.
3. If the user clicks **OK** on the Filter Wizard, the following occurs:
   - The filter is submitted back to the source file, to the designated target cell.
   - The source file is calculated and refreshed (including running Axiom queries set to **Refresh on Manual Refresh**).
   - The form web page is updated to show the latest data.

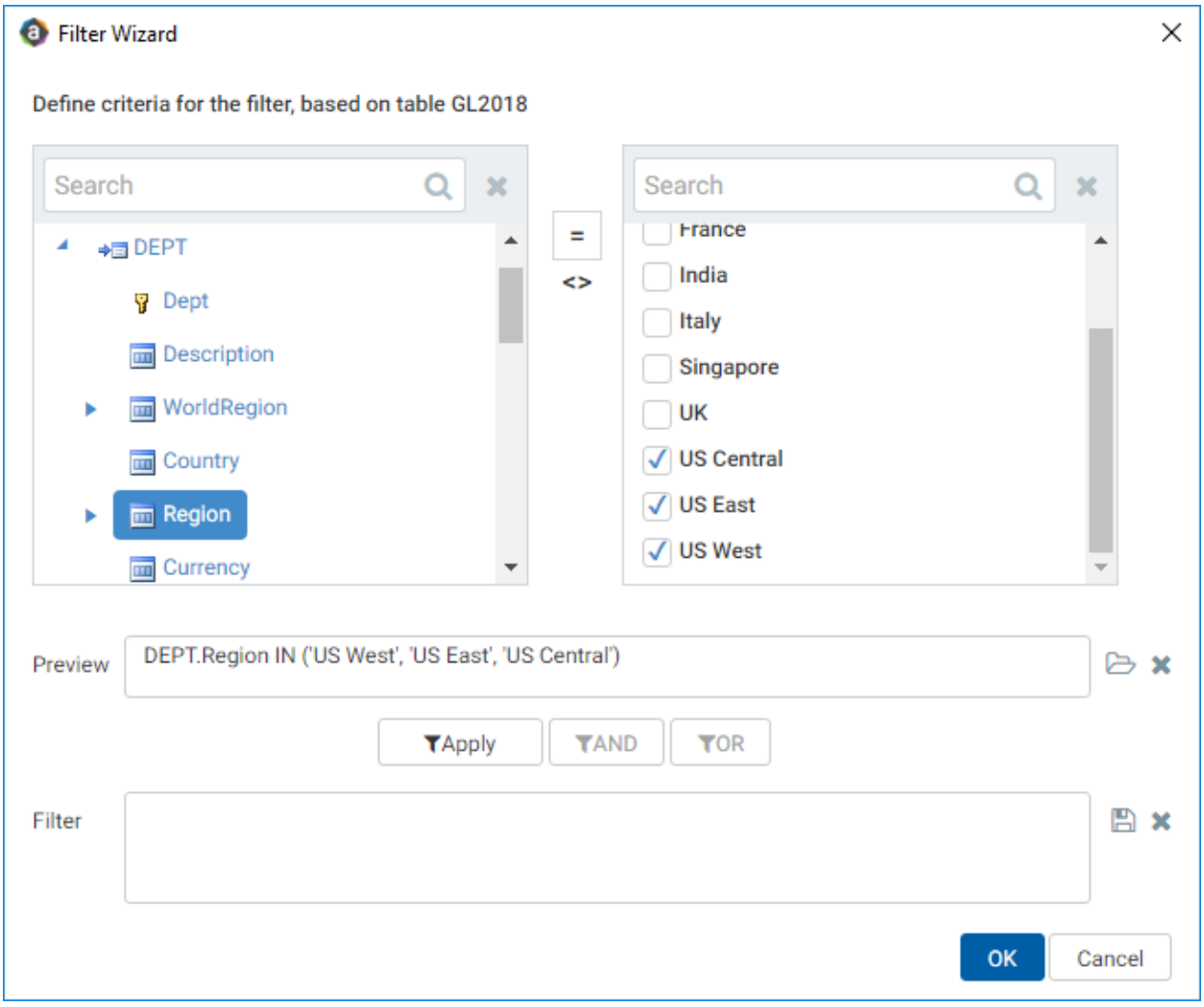
NOTE: Clicking OK in the Filter Wizard does not clear or change the triggering component. The triggering component will still be recorded as the button that opened the Filter Wizard. Therefore if you want to dynamically enable or disable Axiom queries for the data refresh, you can base the formulas on the button with the Filter Wizard command.

4. If the user clicks **Cancel** on the Filter Wizard (or clicks the close button in the top right corner), then the dialog is closed and no further actions occur. The form data is not refreshed and the form web page is not updated. Although interactive values were submitted at the start of the process, you will not see the effects of any changes until the form is updated using another component.

User experience

In Axiom forms, you configure the wizard to use either the Advanced Filter view or the Hierarchy Filter view (known as Simple View in the Desktop Client). It is not possible for the user to switch between views—only the specified view is available in the wizard.

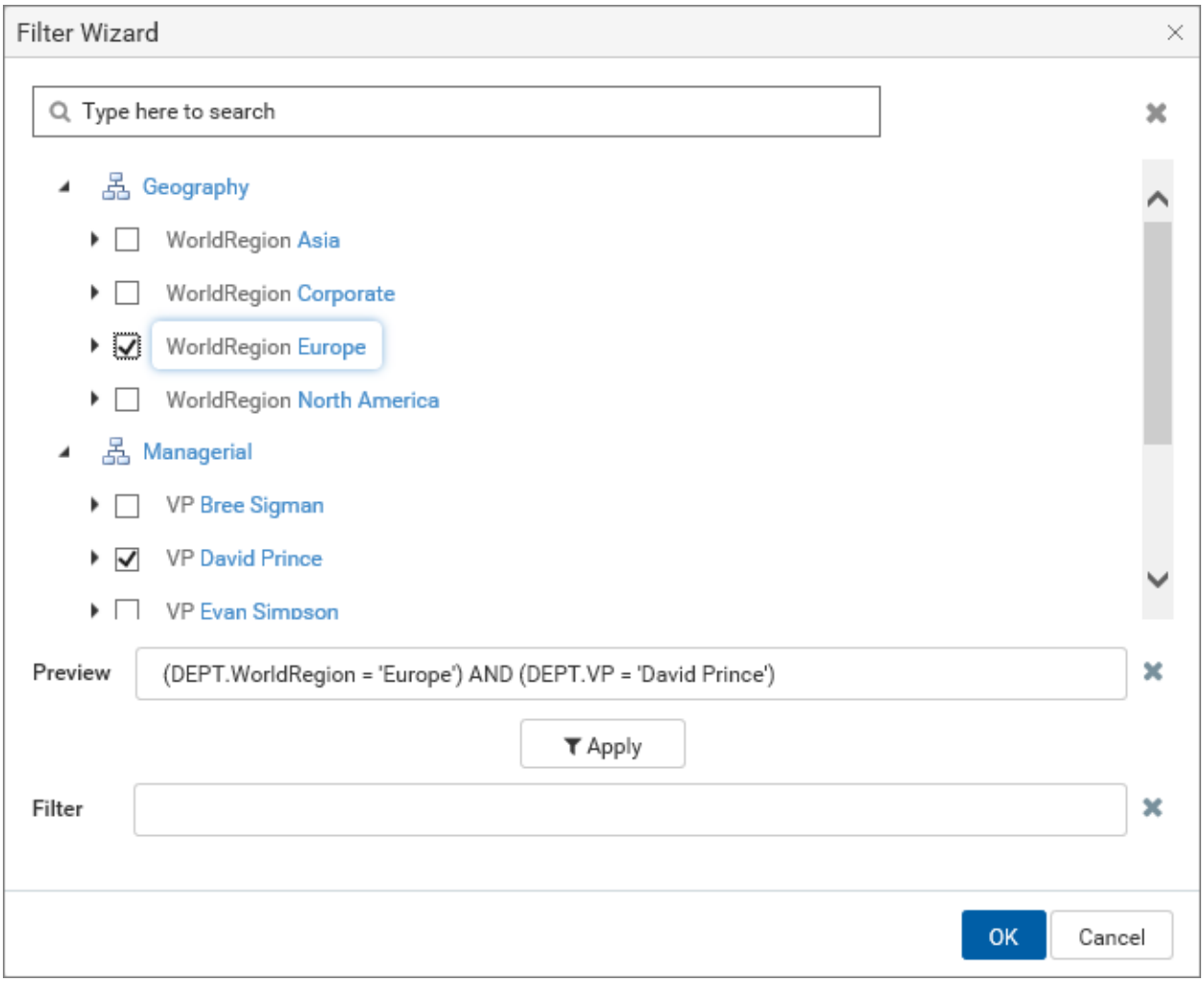
When using Advanced Filter view, users can create a filter based on literal table columns and values in those columns. The dialog shows only the tables listed in the **Tables / Hierarchies** shortcut parameter (as well as their lookup tables). To create a filter, select a table column, select an operator, and select one or more values.



Example Advanced Filter view

In Advanced Filter view, users also have the option to load saved filters from the Filter Library, and save filters to the Filter Library for future use.

When using Hierarchy Filter view, users can create a filter based on selections in one or more hierarchies. The dialog will either show all hierarchies, or only the hierarchies listed in the **Tables / Hierarchies** shortcut parameter. To create a filter, select one or more items. A filter criteria statement is automatically created based on the selected items.



Example Hierarchy Filter view

When using either view, the filter is initially built out in the **Preview** box. Click the **Apply** button underneath the Preview box in order to copy the filter down to the **Filter** box. Clicking the **OK** button then closes the dialog and sends that filter to the form source file. If the OK button is clicked when the Filter box is blank, then a blank filter will be submitted.

When using the Advanced Filter view, compound statements can be created as follows:

- Create one statement, then click **Apply** to move the statement down to the Filter box.
- Create a second statement, then click **OR** or **AND** to combine the second statement with the first statement using the specified operator. Repeat as needed.

When using the Hierarchy Filter view, the Preview and Filter boxes just provide a means to preview the newly created filter as compared to the current filter (if any). The Hierarchy Filter view cannot be used to create compound statements.

NOTE: In both views, it is possible to manually edit the filter criteria statement, which may result in invalid filters. An invalid filter will cause an error in the form if the filter is being applied to data queries. In this case the user must use the Filter Wizard again to define a new, valid filter (including blank for no filter), or reload the form to start over.

If a **Limit Filter Column** is specified with Advanced Filter view, then the dialog changes to a two-step process that allows the user to make selections to generate a limit query statement. In this case, the user experience is the same as when using an AdvancedFilter refresh variable or the ShowFilterWizardDialog function to create a limit query statement.

Design considerations

- If you want the Filter Wizard to start off with a filter, you can enter a filter criteria statement into the designated target cell for the Filter Wizard command. When a user launches the Filter Wizard for the first time, this starting value will display in the Filter box. The user can then create a new filter and overwrite your starting filter.
- The Filter Wizard allows the user to submit a blank filter. Make sure that your file is configured to handle a blank result without error.
- The list of available tables or hierarchies in the wizard is not automatically filtered based on the queries in the file. Depending on how the filter is being used in the form source file, and depending on the tables or hierarchies available in the dialog, the user may create a filter that has valid syntax but is still invalid in the context of its eventual use case. You should define the Tables / Hierarchies parameter as narrowly as possible to help avoid the possibility of query errors due to invalid filters.

Displaying custom help text for Axiom forms

You can display custom help text within an Axiom form, to provide context-sensitive assistance to users while they are working in the form. For example, you might want to provide explanation of certain terms or data displayed in the form, or provide instruction on how to use the form.

Using custom help text is a two-step process:

- Using the Form Help Admin page in the Web Client, you define the help text to display and assign it a unique code. This help text is then stored in the Axiom Software database, where it can be referenced in the form using the code. For more information, see Managing custom help codes.

- Within the form, you configure it to display the help text using either the Form Help component or a form-level property.
    - The form-level property **Help Code** should be used when you want to display help text associated with the entire form. This associates the help icon in the Web Client task bar with the specified code. For more information, see Associating an Axiom form with a help code.
    - The Form Help component should be used when you want to display help text associated with a particular area or feature within the form. For example, you may have multiple sections in a form and you want to provide context-sensitive help for each section. You can place the Form Help component next to the header for each section to display help for each section. You can also use the `[HelpCode]` property for Menu components to associate a Form Help component with different items in a menu. For more information, see Form Help component and Menu component .

When a user is in the form and clicks on a help button that is associated with a form help code, a help panel opens on the right-hand side of the page to display the help text.

Managing custom help codes

You can define custom help text for use within web-enabled files in Axiom Software. This feature is intended to provide custom documentation for individual files—for example, to provide instruction on how to fill out the fields in an input form, or to provide more information on the data and terms shown in a dashboard.

To create custom help text, you define the following:

- A user-defined help code that identifies the help text
- A title for the help text
- Body text to define the instructional or informational text for the form user

Once the help text is defined, it can be used in Axiom forms and web reports, using the following features:

- **Help Code** property at the file level
- Form Help component (Axiom forms only)
- Menu component (Axiom forms only)

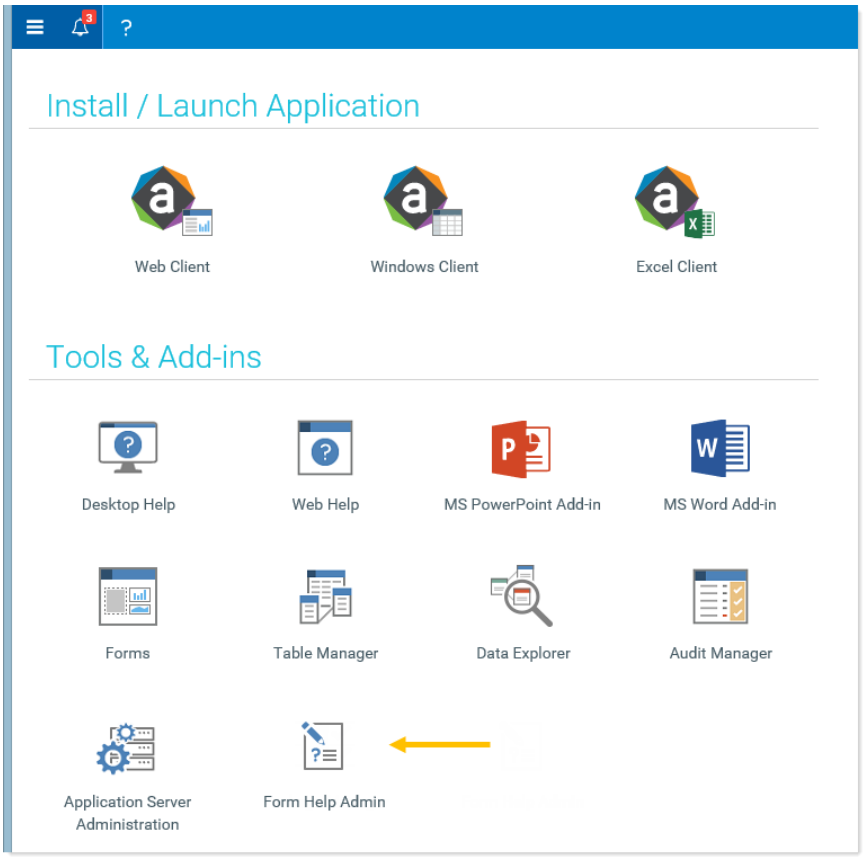
When a user clicks a help icon that is associated with a custom help code, a panel opens along the right-hand side of the page to show the associated help text.

NOTES:

- Only administrators can create, edit, and delete custom help text for Axiom forms.
- The ability to define custom help text is separate from the Axiom Software Help delivered with the software. The form help feature is intended to support form-specific, context-sensitive help for the unique forms in your system. The Form Help component is the only way to display the custom help text.

Accessing the Form Help Admin page

Help codes are managed in the Web Client, on the Form Help Admin page. To access this page, click the **Form Help Admin** icon on the Axiom Software launch page.

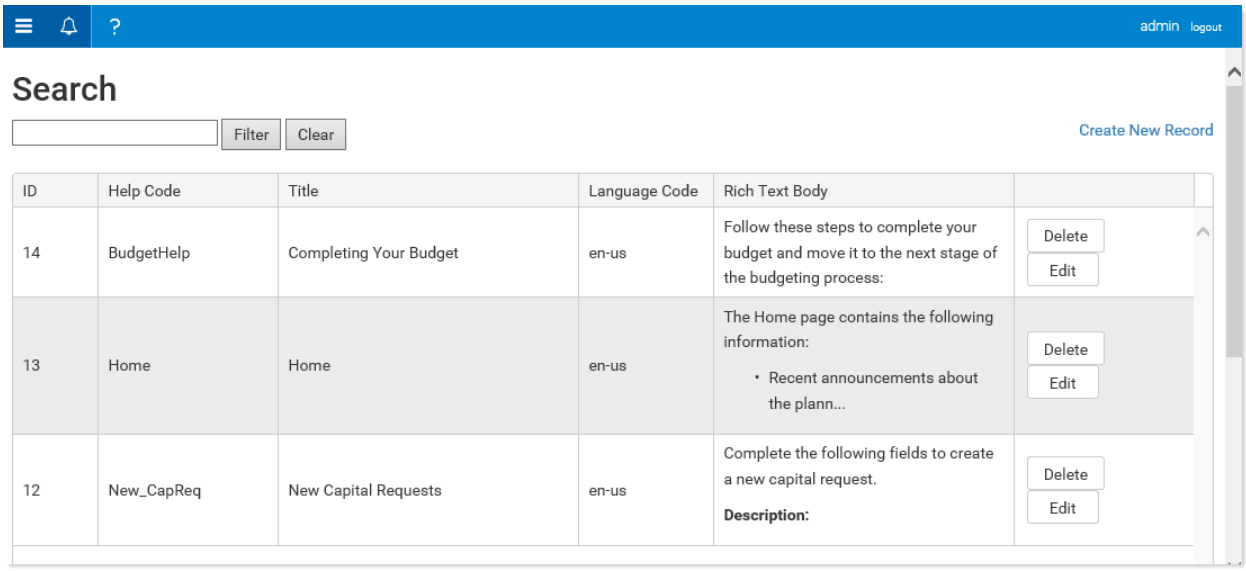


Axiom Software launch page

You can also access the Form Help Admin page directly using the following URL:

Example On-Premise URL	`http://ServerName/Axiom/Formhelp/FormHelpSearch` Where *ServerName* is the name of the Axiom Application Server, and Axiom is the default name of the virtual directory.
Example Cloud System URL	`https://CustomerName.axiom.cloud/FormHelpSearch` Where *CustomerName* is the name of your cloud service system.

The Form Help Admin page displays a grid with all of the help codes defined in your system. If the grid is blank, then no help codes have yet been created. If your system has one or more installed products, then you may see help codes that were created as part of the product installation, for use in files provided by the product.



Example Form Help Admin page

You can use the Search box at the top of the page to filter the grid. The search checks for matches in the following fields: Help Code, Title, and Rich Text Body. This can be helpful to locate a particular help code.

Adding a help code

You can add as many help codes as needed. Each help code must have a unique help code / language code combination.

To add a new help code:

1. In the Form Help Admin page, click **Create New Record**.
2. In the Create New Form Help page, complete the following:

Item	Description
Help Code	The code to identify the help text. This is the code that you will use in Axiom forms and web reports in order to display the help text. The code can be up to 50 characters, and can use numbers, text, spaces, and special characters. When viewing custom help in the help panel, the help code displays in a tooltip when hovering your cursor over the help title.
Title	The title for the help text. This text displays using a title format at the top of the help panel. The title text can be up to 100 characters, and can use numbers, text, spaces, and special characters.
Language Code	The language to associate with this help code, so that users running Axiom Software in that language will see this help text. Currently, there are two available choices: • **en-us** (default): English • **fr-fr**: French If all of your users run Axiom Software in the same language, select that language if it is available. Otherwise, use the default of en-us. If you have a multi-language deployment and you want to define help codes that work with multiple languages, see Using multiple languages with help codes.
Rich Text Body	The body text for the help. You can use the rich text editor to apply formatting to the text. For more information, see Defining the help text in the rich text editor.

3. If you want to see what the help code will look like when it is viewed in a file, click **Preview**. This will show the help text in the same help panel that is used in Axiom forms and web reports.
4. Click **Save** to save the new help code.

The new help code can now be used in Axiom forms and web reports.

Editing a help code

You can edit help codes that your organization has created at any time. Generally speaking, users will see these changes immediately (though, not if the help code is currently open in a panel).

If your system contains help codes that were created as part of a product installation, these topics should not be edited.

To edit a help code:

1. In the Form Help Admin page, locate the help code that you want to edit, and then click **Edit** on that row. You can use the search box at the top of the page to find the code.
2. In the Edit Form Help page, edit any of the help properties as needed. Keep in mind the following:
    - If you change the help code, this will break any references to the code in Axiom forms and web reports. You should not change the help code unless you know that the code is not being used, or you are prepared to manually locate and update all files that use the code.
    - If you change the body text, the changes will be immediately viewable by end users as soon as you click save. If you need to make extensive changes to body text that may take several revisions, and the help code is referenced by files that are actively being used by end users, you may want to make your changes in a new help code. When you are done, you can either edit your files to point to the new code, or delete the old code and give the new topic the old code.
3. If you want to see what the help code will look like when it is viewed in a file, click **Preview**. This will show the help text in the same help panel that is used in Axiom forms.
4. Click **Save** to save your changes.

Deleting a help code

You can delete help codes that your organization has created at any time. Keep in mind that if you delete a help code that is referenced by an Axiom form or web report, that reference will now cause an error. You should be sure that the help code is no longer needed before deleting it.

If your system contains help codes that were created as part of a product installation, these topics should not be deleted.

To delete a help code:

- In the Form Help Admin page, locate the help code that you want to delete, and then click **Delete** on that row. You can use the search box at the top of the page to find the code.

The help code is deleted. There is no way to undo this action.

Using multiple languages with help codes

You can configure help codes to display in multiple languages, so that users see the correct language version for the specified help code. This feature works as follows:

- When a particular help code is opened in the help panel, Axiom Software first looks to see if that help code has an entry with a language code that matches the current language (as determined by the browser's configured language).
- If a match is found for the help code and language code, that help code is displayed. Otherwise, the English (en-en) version is displayed.

For example, if you have users that use both English and French, you can provide help as follows:

- Create one help code with a code of DashboardHelp, language set to en-us, and body text written in English.
- Create another help code with a code of DashboardHelp, language set to fr-fr, and body text written in French.
- Set up the file to use the help code DashboardHelp.

In this example, when a user running an English system views the file, they will see the English version of the help code. And when a user running a French system views the file, they will see the French version of the help code. If users are running a third language, such as Swedish, those users will see the English version of the help code (because there is no match for the specified help code with a Swedish language code).

Defining the help text in the rich text editor

When you create or edit a help code, you can define the body text for the help using the rich text editor. The rich text editor provides basic font formatting, as well as lists and alignment.

If you want to add a symbol or a link to the text, you must edit the HTML directly. Click the **</>** button to open the **View HTML** window. Within this window, you can manually add the following:

Item	Description
Symbols	To display a symbol in the help text, use syntax such as the following: `<span class="fa fa-soccer-ball-o"></span>` Where the text in the class parameter is the symbol name. You can use any symbol that is available for use in the Symbol tag and other areas of Axiom forms. The Symbol Chooser dialog in Axiom forms can be used to find the symbol name.
Hyperlinks	To include a link in the help text, use syntax such as the following: `<a href="http://www.axiomepm.com" target="_blank">Axiom Software</a>` Where the text in the href parameter is the URL, and the text in between the `<a>` tags is the display text. The link opens in a new window using this syntax.

When creating the help text, it is recommended to only use the rich text editor to apply formatting, and only use the View HTML window to add these special items or to troubleshoot formatting issues. Any HTML manually added to the help text is done at your own risk.

Associating an Axiom form with a help code

You can associate an Axiom form with a custom form help code, in order to display form-specific help text when users are in the form.

When a form-level help code is specified, the help icon in the right-hand side of the Web Client task bar opens a help panel to show the associated help text, instead of opening Axiom Software Help.

Example help panel showing file-specific custom help

Users can still open Axiom Software Help as needed by clicking the link at the top of the panel.

The form-level help code can be used by itself, or in conjunction with Form Help components placed within the form itself. For example, you may have overall help text that applies to the entire form, and then context-sensitive help text for specific sections or specific inputs within the form itself.

NOTE: The form-level help code cannot be used with embedded forms. Only the help code for the parent form will be used in this case.

To specify a help code for an Axiom form:

1. From the top of the **Form Assistant** task pane or the **Form Designer** dialog, click **Edit form properties**.
2. In the **Form Properties** dialog, enter a code into the **Help Code** field. The code must match a form help code defined in the Form Help Admin area of the Web Client.

TIP: You can also specify a help code by editing the **HelpCode** field at the top of the Form Control Sheet.

Publishing Axiom Forms

The Axiom form that users interact with is a web representation of the form settings in the source Axiom file. When a user opens the file as an Axiom form, the form web page is automatically generated "on the fly" by the Axiom Application Server based on the form settings in the file. The Axiom form designer does not need to "generate" or "publish" a separate web-enabled file—in this context, "publishing the Axiom form" simply refers to ways in which end users can view the file as a web form.

There are various ways that end users can open form-enabled files as Axiom forms. The exact options depend on which Axiom Client your end users will be using:

- **Web Client**: When using the Web Client, users are already in the native environment for Axiom forms—the web browser. Users can browse the forms available to them and open them directly as forms.
- **Excel Client / Windows Client**: When using one of the desktop clients, forms can be opened as either the source spreadsheet file or as a web form. The type of file and the method of access determines how the file is opened.

In all cases, the user must have at least read-only permission to the source file in order to view it as an Axiom form. If the Axiom form is configured to save data, then the user must have the Allow Save Data permission as well.

Previewing an Axiom form

You can preview an Axiom form to see how it will appear to end users. This is the same behavior as if you had opened the file separately as an Axiom form. The form is fully interactive.

To preview an Axiom form:

- To preview the form within the Axiom Software client (as a tab in the application), then click the **Preview** button on the Form Assistant task pane. You can also use **File Options > Forms > Preview Form** on the Axiom ribbon.
- To preview the form in your browser, click the **Preview** button in the Form Designer. You can also use **File Options > Forms > Preview Form in Browser** on the Axiom ribbon.

In some cases you may want to preview the form in your browser, because that environment provides access to various troubleshooting tools. For more information, see Troubleshooting Axiom forms. These tools are not available when viewing the form as a tab within the Windows Client.

NOTES:

- The source file will automatically be saved when you click **Preview**. If the file is a new file that has not yet been saved, then you must save it to the Axiom file system before you can preview.
- When you preview an Axiom form, a copy of the source file is opened by the Axiom Application Server and this is what is used for the preview (just like when end users view Axiom forms). If you change interactive components in the Axiom form preview, you will not see any values change in the currently opened file—the values are changing in the copy opened by the server.
- If the Axiom form is configured to save data, this save will occur in preview mode—the Axiom form is fully "live" in this context.
- If you are using the Axiom Excel Client with Excel 2013 or 2016, then the form always opens in a browser. There is no option to open the form as a web tab within the application.

Previewing an Axiom form as another user

If you are an administrator and you want to see what the Axiom form will look like when opened by a user with a particular set of rights, you can open the Axiom form as that user for testing purposes. With the form-enabled file open, do the following:

1. On the **Axiom** tab, in the **File Options** group, click **Forms > Preview Form in Browser (as other user)**.

   TIP: The command **Open form in browser (as other user)** is also available by right-clicking form-enabled files in Axiom Explorer or the Explorer task pane, and by right-clicking a form tab in the Excel Client or the Windows Client.

2. In the **Select User** dialog, select the user and then click **OK**.

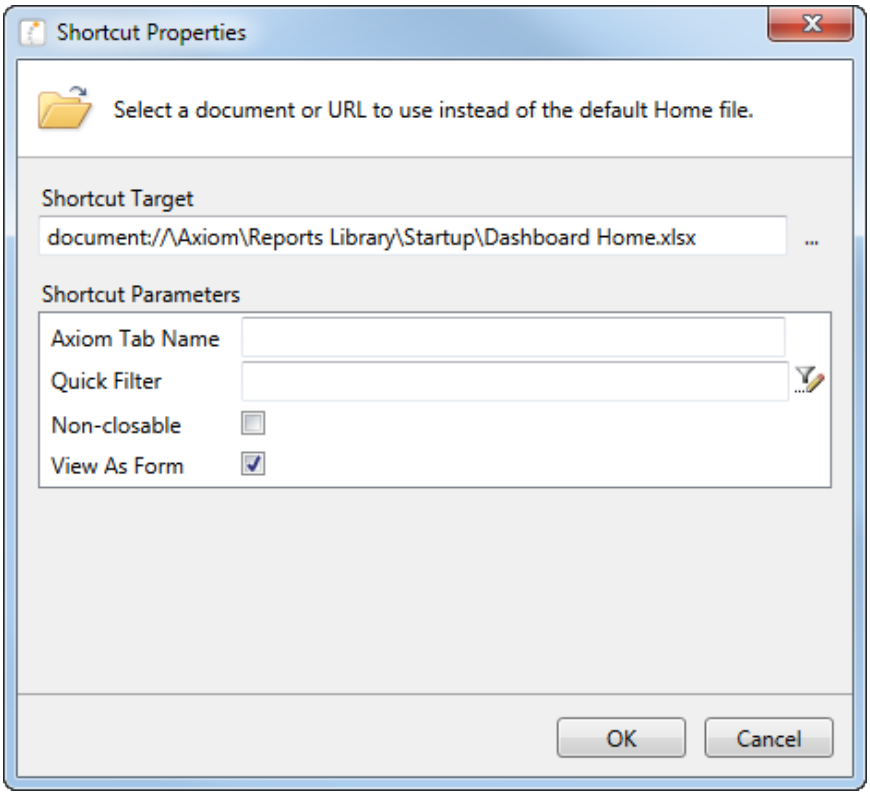
The Axiom form opens in a browser window, and the data in the Axiom form is updated based on the selected user's security permissions. If the selected user does not have rights to view the Axiom form, then an error message will display.

NOTE: If you have the Axiom Web Client open in a browser when you preview the Axiom form as another user, the login for the web page will be changed to that user. You will then need to log out and log back in as yourself.

Using an Axiom form as the Home Page or other startup document

Axiom forms are frequently used as home pages in Axiom Software. For environments where end users primarily interact with Axiom forms, it is common to design the home page as a "forms portal" to provide hyperlinks to the other forms that users need to access, as well as to display other information in a graphical web format.

You can assign an Axiom form as the Home Page (or as an "other startup document") using the **Startup** tab in security. When you set the shortcut to a forms-enabled file, you must enable the **View As Form** option in order to open the file as an Axiom form.



Home Page

If an Axiom form is set as the Home Page (or as the Desktop Client Home Page), then when the user logs into the Excel Client or the Windows Client, the Axiom form will display as the home page in the client session. The Axiom form displays in a web tab within the Desktop Client, in all cases (including Excel 2013 and 2016).

For the Web Client, the Axiom form displays as the user's Home page as follows:

- If the user accesses the base URL for the Axiom Software installation, the form-enabled Home file displays as the user's default page. For example, if a user goes to `https://CustomerName.Axiom.Cloud`, the user's Home page will display.

  NOTE: This behavior only occurs if the Web Client Container is enabled for the form. If the Web Client Container is not enabled for the form, then the launch page displays instead.

- If the user accesses the form Home page for the Axiom Software installation. For example, if a user goes to `https://CustomerName.Axiom.Cloud/forms/home`, the user's Home page will display.

Other startup documents

If an Axiom form is set as an "other" startup document, then when the user opens the Excel Client or the Windows Client, the Axiom form will open in addition to the home page. For the Windows Client only, you can optionally define the **Axiom Tab Name** in order to open the form within the client session instead of in the browser. In all other cases the form opens within the browser.

Other startup documents are ignored when a user logs into the Web Client; only the Home Page assignment is honored.

Accessing Axiom forms using the Desktop Client

The treatment of form-enabled files in the Desktop Client (Excel Client or Windows Client) depends on the type of file, the method of access, and your Excel version.

NOTE: This topic discusses ways to open Axiom forms directly in the Excel Client and the Windows Client. You can also open Axiom forms via hyperlinks within another forms (such as a home page form). For more information on setting up an Axiom form to contain hyperlinks to other Axiom forms, see Hyperlinking to other files in an Axiom form.

Form-enabled plan files

If a plan file is form-enabled, then it will automatically open as a web form using all normal methods of plan file access—such as Open Plan Files, the Process task pane, and process notification emails. You do not need to specially configure this behavior.

If you are an administrator or other power user who needs to open a form-enabled plan file as the source spreadsheet file, then you can right-click the file in Open Plan Files and choose **Open as Spreadsheet**.

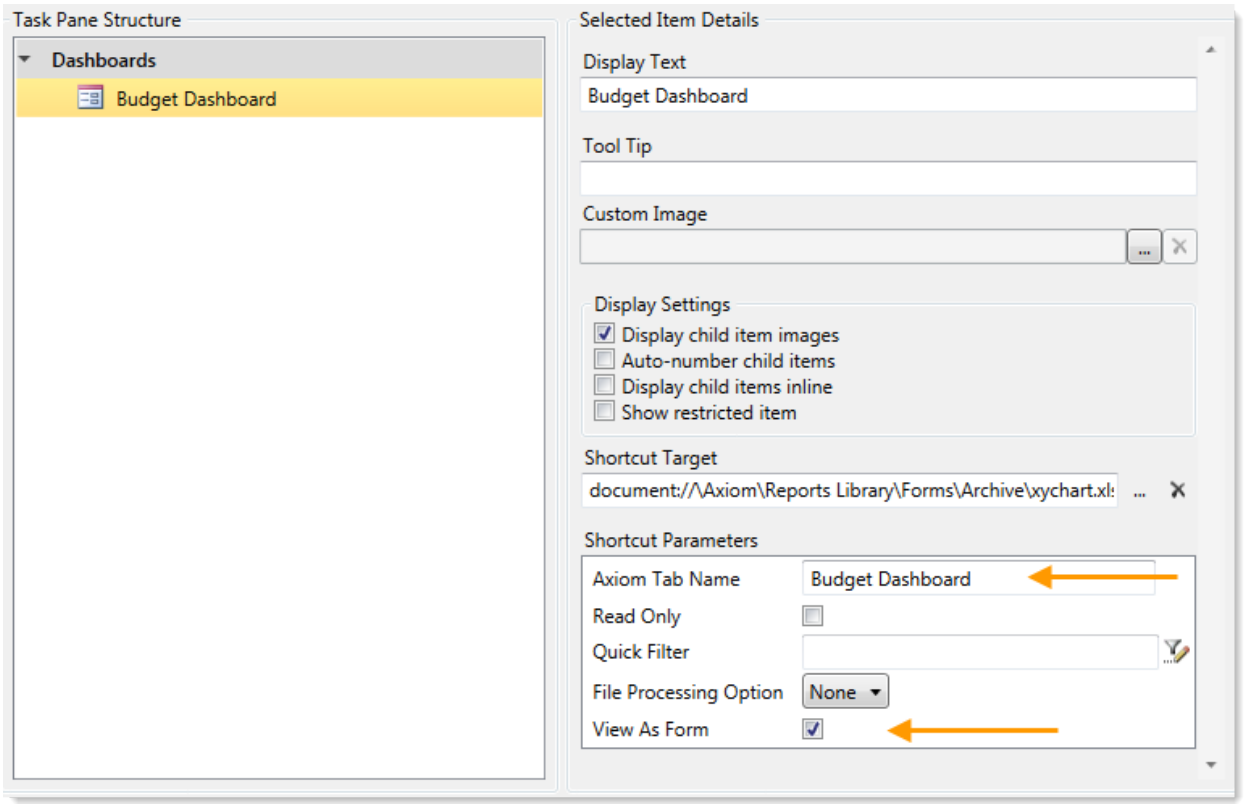
If you are using the Windows Client, then any form-enabled plan files opened within the Desktop Client will open as web tabs within the application. If you are using the Excel Client, then any form-enabled plan files opened within the Desktop Client will open in the user's browser.

Form-enabled reports

Generally speaking, if you want a form-enabled report to open as a web form, you must specially configure this access. When a form-enabled report is opened directly from the Reports Library, it will always open as the source spreadsheet file.

To allow end users to open the report as a web form, you can set up a custom task pane to open the file. You can link to the source report and then select the **View As Form** option. When the user double-clicks the item in the task pane, the report will open as an Axiom form.

If you want the Axiom form to open as a tab within Axiom Software, then you should also define the **Axiom Tab Name**; otherwise the Axiom form will open in the user's browser. This only applies to the Windows Client. When using the Excel Client, the form always opens in the user's browser.



Example task pane configured to open an Axiom form

Opening Axiom forms using the Axiom Web Client

Users can open their Axiom forms using the Axiom Web Client in a browser. The following is an overview of how this access works.

NOTE: An Axiom Software app is also available for Apple® iPad® to view forms via the Web Client. The app provides an alternative to using the browser.

Using a form Home page to access other forms

If users are assigned a form-enabled Home file, they should use the following URL to view forms in the Web Client:

Example On-Premise URL	`http://`*`ServerName`*`/Axiom/Forms/Home` Where *ServerName* is the name of the Axiom Application Server, and Axiom is the default name of the virtual directory.
Example Cloud System URL	`https://`*`CustomerName`*`.axiom.cloud/Forms/Home` Where *CustomerName* is the name of your cloud service system.

Going to this URL will display the user's form Home page automatically. The Home page can be set up with links to other Axiom forms so that users can access the forms they need. Users can also use the Navigation panel to open other forms (see the following section).

NOTE: If the Web Client Container is enabled for the form Home file, then this Home page will display automatically when the user navigates to the base URL for the Axiom Software installation. In this case, it is not necessary to provide users with the form-specific URL listed in the previous table. However, if the Web Client container is not enabled for the form Home file, then users must use the form-specific URL in order to start at the Home page.

Using the Navigation panel to access forms

If the Web Client Container is enabled for your forms, then users can access other forms by using links in the Navigation panel. By default, this panel contains links to all of the form-enabled files within the Reports Library that the user has access to. You can optionally customize the contents of this panel to display different links, or to organize the links differently.

The Navigation panel can also display links that are relevant to the current form only, so that these links only display when the user has the form open.

Use of the Navigation panel can reduce or eliminate the need to include links within your forms. This can free up content space within forms, and also make it easier to maintain links.

For more information on setting up the Navigation panel, see Defining navigation links for the Web Client Navigation panel.

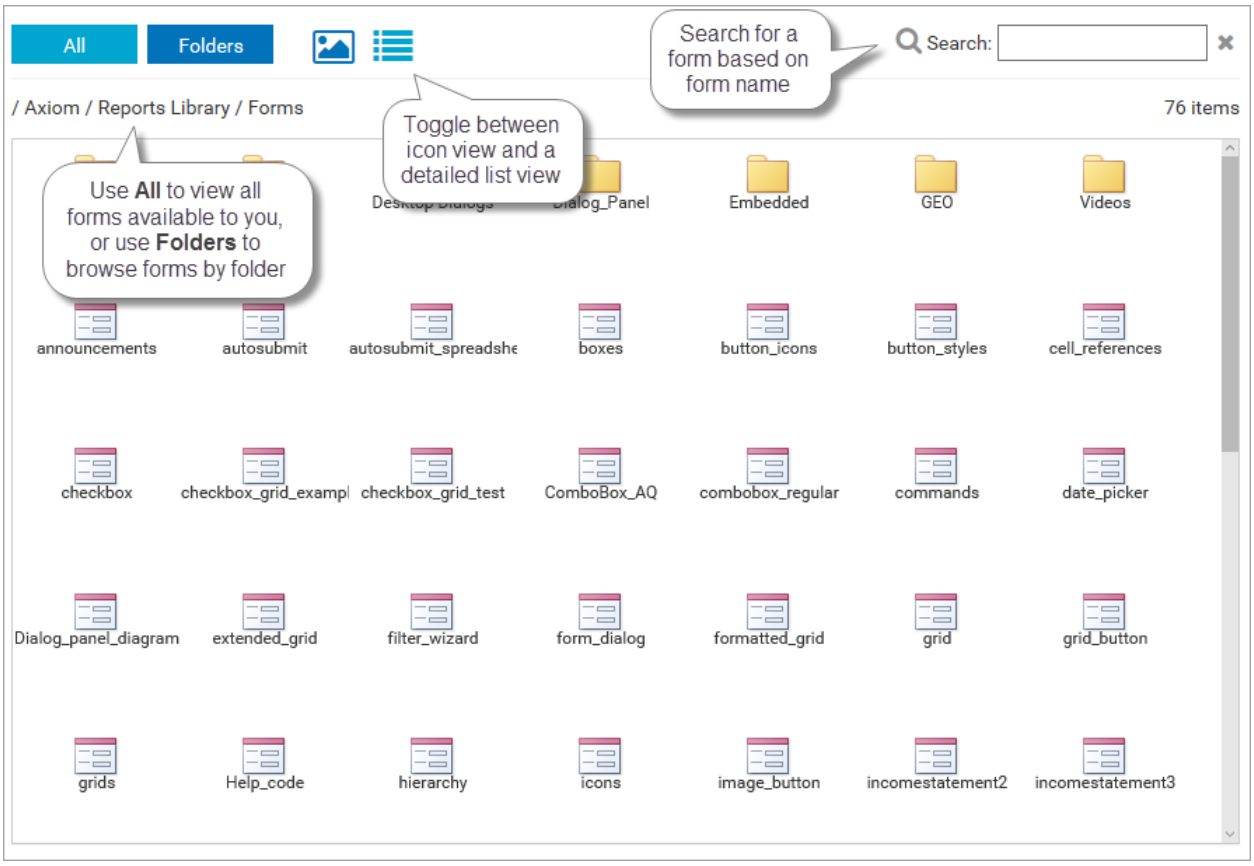
Browsing available forms

If users do not have a form-enabled Home file, they can use the following URL to browse and open forms in the Web Client:

Example On-Premise URL	`http://ServerName/Axiom/Forms` Where *ServerName* is the name of the Axiom Application Server, and Axiom is the default name of the virtual directory.
Example Cloud System URL	`https://CustomerName.axiom.cloud/Forms` Where *CustomerName* is the name of your cloud service system.

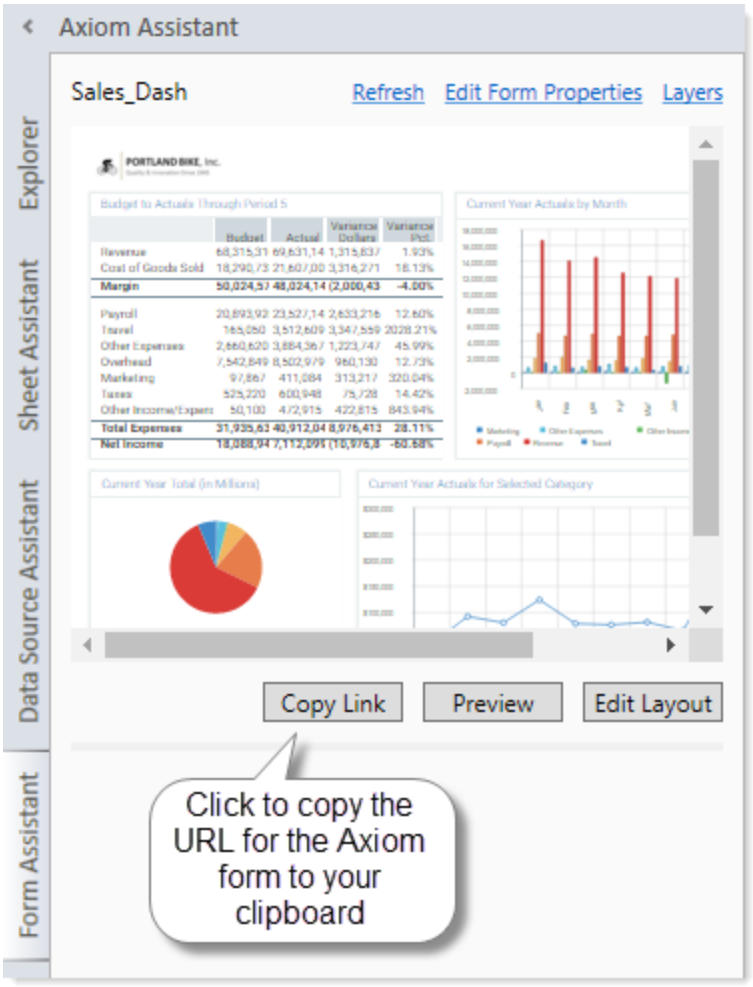
This area can also be accessed from the Axiom launch page, by clicking the **Forms** icon.

This Forms Explorer page displays a list of all form-enabled files that the user has permission to access, located within My Documents, the Reports Library, or file groups. Users can browse by folder, or they can use the search box to find a specific form.



Creating a hyperlink to an Axiom form

You can provide a user with a URL that opens the file as an Axiom form in the Axiom Web Client. To do this, use the **Copy Link** button in the Form Assistant task pane. This will generate a link for the Axiom form and then copy it to your computer's clipboard. You can then paste the link as needed—for example, in an email, or a document, or a corporate portal, etc.



For example, after clicking Copy Link, you could go to an open email and then use CTRL+V to paste in a URL similar to the following:

`http://servername/Axiom/forms/5tJwo--aAMoYdcVYTZ-1Sg__`

When a user clicks an Axiom form hyperlink, the form opens in the user's browser using the Axiom Web Client. The user must be validated by Axiom Software in order to open the form.

Saving a snapshot of an Axiom form

You can save a snapshot copy of an Axiom form, so that you can view the form "offline" without needing to be logged into Axiom Software. This is similar to the snapshot feature available for Axiom files in the Desktop Client, except that in this case the snapshot is of the web page rendition of the file, not the source workbook.

Taking a snapshot

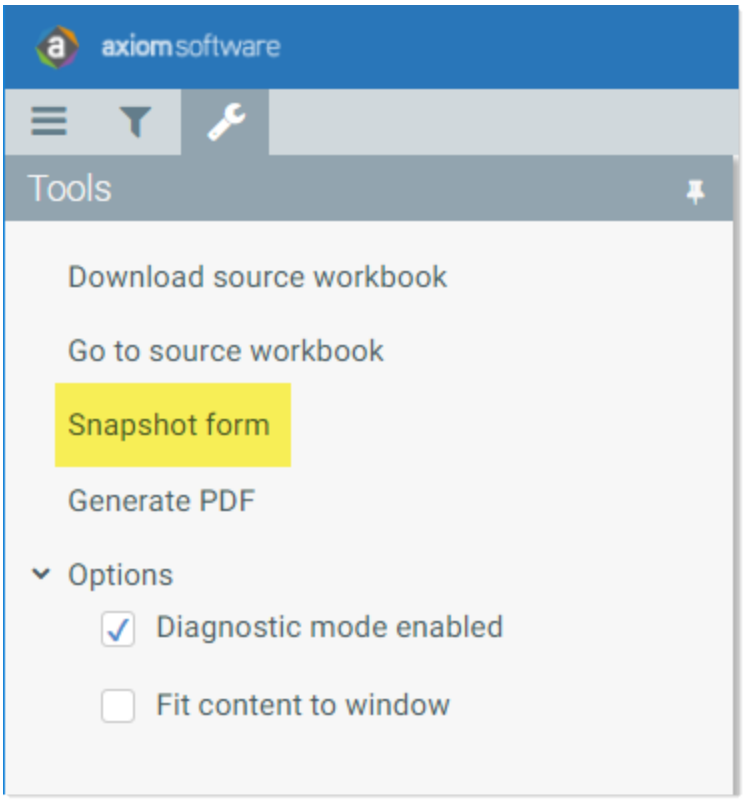
Before taking a snapshot copy of an Axiom form, keep in mind the following known limitations:

- The data displayed in the Axiom form is frozen as of the point when the copy is made. Basically, whatever you see in the form when the snapshot is taken is what subsequent viewers will see when they open the disconnected form.
- Interactive components are not supported in snapshot copies. If you attempt to use an interactive component such as a combo box or a button, a message will inform you that the feature is unavailable.
- Snapshots are not supported for Wizard Panel components. The wizard elements will not be included in the snapshot.

The snapshot feature is only available when the Axiom form is open in the Web Client browser.

To take a snapshot copy of an Axiom form:

- From the Tools menu (the wrench icon), click **Snapshot form**.



Your browser will prompt you to save the snapshot copy. The snapshot uses MHTML format (.MHT), which is a format that combines graphics and other web features along with HTML code into a single file.

NOTE: This feature is not supported when using the Web Client on a mobile device.

Viewing a snapshot

Although you can create the snapshot copy within any supported browser for Axiom forms, only Microsoft Internet Explorer (not Edge) has native support for viewing an MHT file. If you are using other supported browsers such as Apple Safari or Google Chrome, you may need to locate and install a third-party add-in to view the file.

NOTE: If you are using Internet Explorer 11 to view the snapshot copy, you may need to refresh the file after opening in order to view the contents.

When viewing a snapshot copy, the legacy "curtain" menu is available so that you can reconnect to the source form if desired. Hover your cursor over the top of the form, then click **Tools > Reconnect to source form**. You can use this command to connect to Axiom Software and open the "live" version of the file (the file from which the snapshot was taken).

You can also view the snapshot copy by embedding it in a PowerPoint slide using the Axiom add-in for PowerPoint. In this environment the snapshot can also be updated for the latest data as needed.

Printing an Axiom form

There are two ways that you can print an Axiom form. You can use the native print functionality of your browser to print the form, or you can generate a PDF copy of the form and then print the PDF.

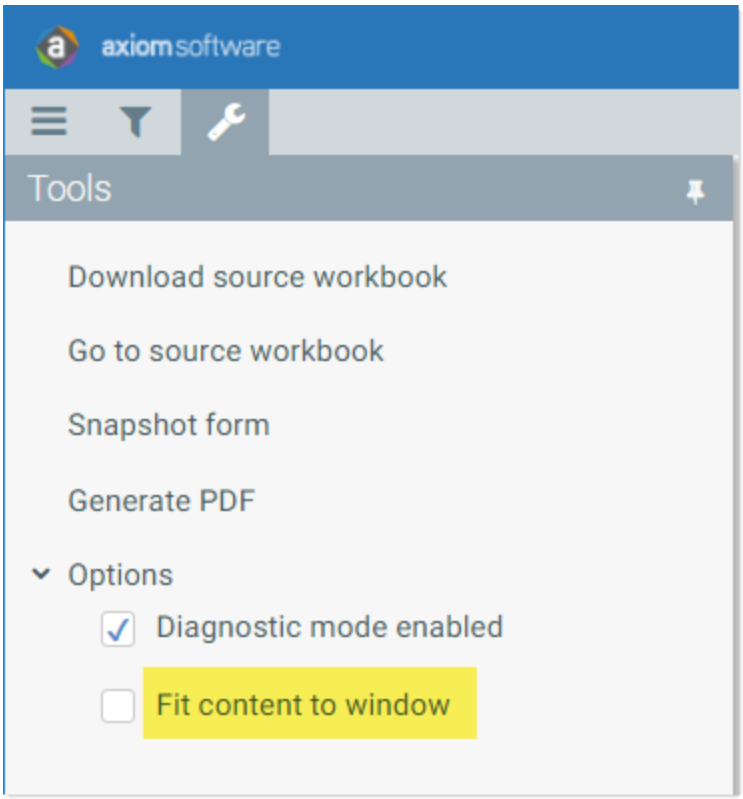
Using browser print functionality

If you print from the browser, then the specific printing procedure and features will depend on the browser that you are currently using. However, keep in mind the following tips:

- Use your browser's Print Preview functionality first to see what the printout will look like, and then adjust from there.
- Many Axiom form designs are better suited for printing in Landscape orientation rather than Portrait.

- If the form does not fit nicely within the page, try enabling or disabling **Fit content to window** and then try Print Preview again.

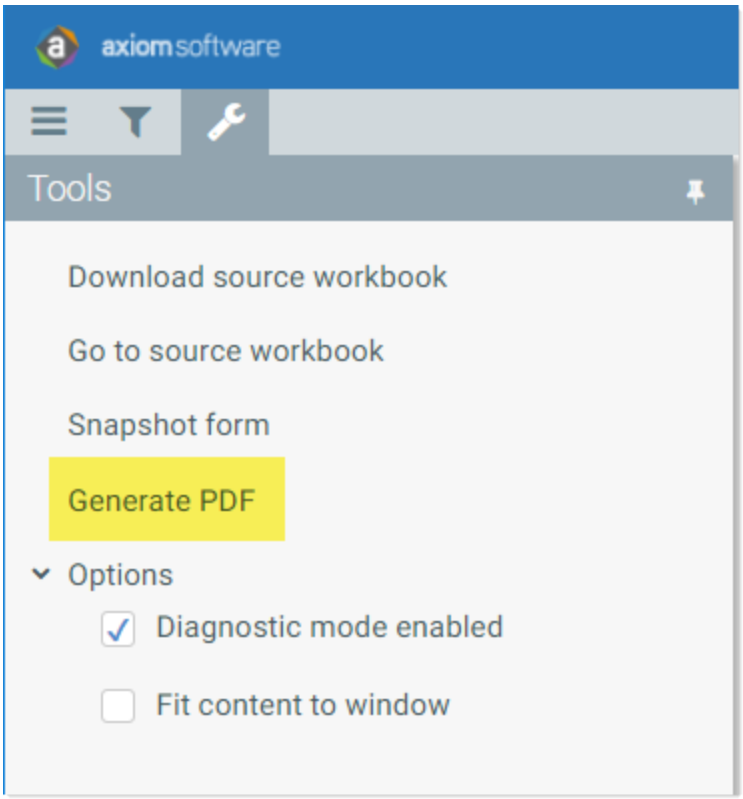
  From the Tools menu (the wrench icon), click **Options > Fit content to window** to select or clear the check box. This option affects forms differently depending on the form configuration. It will not be useful for all forms.



If the Axiom form is opened as a web tab in the Excel Client or the Windows Client, you must open it within a browser in order to gain access to the browser print functionality. You can right-click the file tab and then click **Open in browser**.

Generating a PDF for printing

If the Axiom form is open in a browser, then you can generate a PDF from the Web Client task bar. From the Tools menu (the wrench icon), click **Generate PDF**.



The Axiom form may also be set up with a special button or link to generate the PDF. This could be present within the form as a hyperlink labeled something like "Print this form" or "Create PDF for printing," or as an image (like a printer icon). This option can be used regardless of where the Axiom form is opened.

When using the PDF print option, the form can be specially configured for printing. For example, certain components may be shown or hidden in the print copy, and formatted grids can be automatically extended to show all rows. For more information on configuring a form for printing via PDF, see Configuring an Axiom form for printing (Desktop Client Help).

After a PDF has been generated for the form, it will open in your browser. The specific printing procedures and features will depend on the browser used and your PDF reader.

Custom Dialogs and Task Panes in the Desktop Client

Axiom forms can be used as custom dialogs and task panes within the Axiom Software Desktop Client (Excel Client and Windows Client). This extends the ability to customize the Axiom Software user interface by leveraging the design flexibility of Axiom forms. Example use cases include using Axiom forms as refresh forms, associated task panes, or as generic dialogs for any customized purpose.

When using an Axiom form as a dialog or a task pane, certain specialized functionality is available to provide interaction between the Axiom form and the active client spreadsheet:

- The form dialog / task pane has the ability to pass selected values from the form to the active client spreadsheet. This is accomplished by use of the "form state" feature, which stores selected values in memory. The values stored in the form state memory can then be passed from the Axiom form to the active client spreadsheet. Form state is only for use in this particular context, and only when the form needs to interact with the active client spreadsheet.
- The form dialog / task pane has the ability to execute certain commands on the active client spreadsheet. For example, the form can be used to perform goal seek calculations in the active client spreadsheet, or to process action codes in the active client spreadsheet.

You can also use Axiom forms as dialogs that do not interact with the active client spreadsheet ("stand-alone" dialogs). For example, you could launch a form dialog from a custom ribbon tab or task pane in order to gather some inputs from a user and then save those inputs directly to the database. In this case the form has no association or interaction with the active client spreadsheet.

NOTE: This section discusses how to create an entire form to be used as a custom dialog or task pane within the Desktop Client. If instead you are looking for information on how to create a dialog to be displayed within an Axiom form, see Dialog Panel component.

Using an Axiom form as a custom dialog

You can use an Axiom form as a dialog in the Axiom Software Desktop Client (Excel Client or Windows Client). The dialog can be used for any task that can be performed using Axiom form features.

For example, imagine that you want certain users to be able to add new items to a reference table. Instead of granting them access to Open Table in Spreadsheet or to a save-to-database report, you can

create an Axiom form that allows users to input the necessary information and then save it to the reference table. The user could launch the Axiom form dialog from a ribbon tab or a task pane, or from within an Axiom spreadsheet file. Using a dialog instead of a spreadsheet file provides the user with a more controlled and guided environment for performing the task.

Axiom form dialogs can interact with the active client spreadsheet using "form state" values and various commands, or they can operate as "stand-alone" dialogs that have no relationship to any open files. In the case of a stand-alone dialog, the interactive elements of the form are set up just like a standard form, with no use of form state. Form state should only be used if the form dialog will be launched in a way where the active client spreadsheet file is always known. For example, you can use the ShowFormDialog function to launch the dialog from within a particular file, thereby guaranteeing that any client-side actions performed by the form will be performed on that known file.

NOTE: This section discusses how to create an entire form to be used as a custom dialog within the Desktop Client. If instead you are looking for information on how to create a dialog to be displayed within an Axiom form, see Dialog Panel component.

Designing an Axiom form to be used as a dialog

The Axiom form to be used as the dialog should be configured as follows:

- Set the **Canvas Size** of the form (width x height) to a dialog-appropriate size. When a form is used as a dialog, the canvas size sets the window size of the dialog.

  A good starting point is 700 x 400. To help you design the form to fit the target canvas size, make sure to enable **Options > Show form canvas area** in the Form Designer. If the contents of the form exceed the specified canvas size, scroll bars will be present in the dialog.

- It is recommended to define a form **Title**. This title will be used as the dialog title. Otherwise, the file name of the Axiom form will be used.

- If you want the dialog to interact with the active client spreadsheet, then you can configure the file to use form state and/or various commands that execute on the active spreadsheet file.
  - For more information on using form state, see Passing values between an Axiom form and the active client spreadsheet (form state).
  - For more information on available commands, see Executing commands on the active client spreadsheet from an Axiom form.

- To allow users to close the dialog, you can use the Close Dialog command on a Button component. Users can also close the dialog by clicking the X button on the dialog window. For more information, see Configuring close options for a form dialog.

The form size and title can be set in the Form Properties dialog. To open this dialog, click **Edit Form Properties** at the top of the Form Assistant or the Form Designer.

While you are designing the form, you can use **Forms > Preview Form as Dialog** to see the form as it will display when it is launched as a refresh form. This may assist you in determining the appropriate form size.

Launching the dialog

There are two ways that you can launch a form as a dialog within the Excel Client or the Windows Client:

- **Show Form Dialog command**: This command can be used to open the dialog from a custom ribbon tab or task pane. For more information on using commands in ribbon tabs and task panes, see the *System Administration Guide*.

  In general, this approach should only be used with "stand-alone" dialogs that do not interact with the active client spreadsheet. An exception would be if the command is used in a task pane that is associated with a specific file—in this case you could use the task pane to launch a form dialog that is designed to be used on that associated file.

- **ShowFormDialog function**: This function can be used to open the dialog from within a particular file. For more information, see the *Axiom Function Reference*.

  This approach can be used to launch any type of form dialog—stand-alone dialogs or dialogs that interact with the current file.

The user must have at least read-only access to the designated form in order to open it using either of the above options. If the user does not have permission to the file, an error message will display when the user attempts to open the form. When configuring user security permissions for the form, you can optionally clear the **Show in Explorer** option if the user only needs to access the file via indirect methods such as Show Form Dialog. The form will not display when the user browses file libraries.

When using either of these features, you have the option to set certain form state values and pass them into the target form. This is most useful when using the function. For example, you may want to use the ShowFormDialog function in a calc method, to launch a dialog that shows more details about the current row. In order to filter the dialog to show only the data for the current row, you must pass in form state values that identify the current row (such as the account number for the row). By using a parameter in the ShowFormDialog function, you can specify the account for the current row and pass that into the form dialog when it is launched.

Dialog behavior

When the dialog opens, it is "modal". This means that while the dialog is open, users cannot interact with the currently active spreadsheet file or with any ribbon tab or task pane in Axiom Software. The user must close the dialog before they can return to using the main application.

The user can close the window using the **X** button in the top right corner. You can also design the Axiom form with a button that uses the Close command. For more information, see Configuring close options for a form dialog.

If the dialog contains hyperlinks to documents that will not be opened in the current window, then the dialog automatically closes before opening the document.

Using an Axiom form as a refresh form

You can use an Axiom form as a refresh form, as an alternative to defining refresh variables. This works as follows:

- You design a separate Axiom form to display as the refresh form dialog. This form collects the user's inputs and stores them using the "form state" feature.
- In the file where you want the refresh form to display, you use a special tag to tell Axiom Software to open the Axiom form as a dialog when the file is refreshed. You set up the file to read the desired values from form state, and then use those values to impact the data refresh.

Generally speaking, using an Axiom form as a refresh form requires more setup than refresh variables, but it is also more flexible. For more information on the differences between the two features, see the discussion on refresh forms in the *Axiom File Setup Guide*.

NOTE: This section discusses how to create an entire form to be used as a refresh form within the Desktop Client. If instead you are looking for information on how to create a dialog to be displayed within an Axiom form, see Dialog Panel component.

Designing an Axiom form to be used as a refresh form

The Axiom form to be used as the refresh form should be configured as follows:

- The form must be set up to store one or more values in form state memory. These values are the values that you want to send to the active client spreadsheet to affect the refresh of that file. For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

- The form must use a button that is configured with the following commands: **Apply Form State** and **Close Dialog**.
    - The Apply Form State command sends the updated form state values from the Axiom form to the active spreadsheet file. Note that the **Refresh mode** shortcut parameter for the command is ignored when the form is used as a refresh form. Instead, the level of refresh performed is the level selected by the user when they triggered the refresh from the active spreadsheet file. For example, if the user selected to refresh the active sheet only, then that is the level of refresh that will occur.
    - The **Processing Step** shortcut parameter for the Close Dialog command must be set to **Active Client Spreadsheet - After Processing**.
    - The order of the commands does not matter; the Close Dialog command allows for executing other commands at the After Processing step.
    - Both commands must be on the same button in order to enable the "auto refresh" behavior of the refresh form.
- The form can optionally use a second button that is also configured with the Close Dialog command, this time with the **Processing Step** set to **Active Client Spreadsheet - Before Processing**. The user can click this button to cancel out of the dialog and cancel the refresh. If no cancel button is provided on the form, then the user can click the X button on the dialog window to close the dialog without refreshing.
- Set the **Canvas Size** of the form (width x height) to a dialog-appropriate size. When a form is used as a dialog, the canvas size sets the window size of the dialog.

  A good starting point is 700 x 400. To help you design the form to fit the target canvas size, make sure to enable **Options > Show form canvas area** in the Form Designer. If the contents of the form exceed the specified canvas size, scroll bars will be present in the dialog.
- It is recommended to define a form **Title**. This title will be used as the dialog title. Otherwise, the file name of the Axiom form will be used.

The form size and title can be set in the Form Properties dialog. To open this dialog, click **Edit Form Properties** at the top of the Form Assistant or the Form Designer.

While you are designing the form, you can use **Forms > Preview Form as Dialog** to see the form as it will display when it is launched as a refresh form. This may assist you in determining the appropriate form size.

When deciding where to save the file and what to name it, keep in mind that another file will require the continued presence of this file. If the Axiom form is deleted, moved, or renamed, then the file that is configured to use the Axiom form as a refresh form will no longer work. You may want to name the file so that the relationship between the two files is clear, or save it to a designated folder that holds required "supporting" files such as this one.

Enabling the refresh form for the Axiom file

To launch an Axiom form as a refresh form when an Axiom file is refreshed, place the following tag in a cell within the first 500 rows of a sheet:

`[RefreshDialog;PathToFile]`

For best results, the tag should be placed on the same sheet where the data query is located. This ensures that the refresh form will display if the user chooses to refresh the current sheet only. If it is placed on a different sheet, then the refresh form will only display when the entire workbook is refreshed.

You can manually type the tag, or you can right-click the cell and then select **Axiom Wizards > Insert Refresh Dialog**. This wizard prompts you to select the Axiom form to use as the refresh form, and then places the tag in the cell with the full path to the selected file (using document shortcut syntax). If you manually type the tag, you can enter the path as a normal Axiom file path or using document shortcut syntax.

The next time you open the document after saving, the file parameter will be automatically converted into a system-managed document shortcut (you can tell the difference by the presence of a `_tid` parameter on the end of the shortcut). This is to make the file reference "repairable" in cases where the file is renamed or moved. If you need to change the entry to point to a different file, simply enter the path or document shortcut as you would have originally (or use the wizard again), and the path will be converted again when you save the file. Note that if the tag is a result of a formula instead of directly within the cell, then the conversion will not occur and the file reference will not be repairable.

The presence of the RefreshDialog tag signals to Axiom Software that you want to open an Axiom form as a refresh form. The file path tells Axiom which Axiom form to open. For example:

```
[RefreshDialog;document://\Axiom\SystemFolderName_
ReportsLibrary\Forms\Refresh Form.xlsx?_tid=5599]
```

When the current file is refreshed, Axiom will open the file `Refresh Form.xlsx` as an Axiom form within a dialog window.

NOTE: Users must have at least read-only access to the designated RefreshDialog file in order to open it as a refresh form. Simply having access to the file that calls the refresh form will not automatically grant access to the RefreshDialog file.

Using GetFormState to return values selected in the refresh form

As the user interacts with the refresh form dialog, their inputs and selections are stored in form state memory. When the user clicks the **Apply Form State** button in the dialog, those form state values are passed from the Axiom form to the spreadsheet file. The spreadsheet file must be set up to read these values using the GetFormState function. These values can then be used to impact the data refresh in some way, such as to define the filter an Axiom query.

For more information, see Passing values between an Axiom form and the active client spreadsheet (form state).

Configuring when the refresh form displays

On the Control Sheet, review the **Refresh Forms Run Behavior** setting (under **Data/Zero Options**) and configure it as desired. By default, it is set to **OnManualRefreshOnly**, which means the refresh form will only display when a user manually initiates a refresh in the file. Refreshes that occur automatically when a file is opened will not display the refresh form. The available options are as follows:

- **Off**: Refresh forms are disabled and do not display when the file is refreshed.
- **OnManualRefreshOnly** (default): Refresh forms only display when the file is manually refreshed (by clicking **Refresh**).
- **OnOpenOnly**: Refresh forms only display when the file is refreshed on open. This occurs if an Axiom query is set to **Refresh data on file open**, or if the workbook is set to **Refresh all Axiom functions on open**.
- **OnManualRefreshAndOpen**: Refresh forms display when the file is refreshed on open, and when the file is manually refreshed.

Refresh forms are always ignored when the file is refreshed by an automated process, such as Process Plan Files, File Processing, or by any Scheduler task that opens and refreshes the file.

Using an Axiom form as a task pane

Axiom forms can be used as task panes in the Axiom Software Excel Client or Windows Client. When used as a task pane, the Axiom form opens within the Axiom Assistant area, and can be used to impact the active spreadsheet file.

There are two ways that an Axiom form can be used as a task pane:

- You can associate the Axiom form with a particular Axiom file using the **Associated Task Pane** setting on the Control Sheet. When the spreadsheet file is opened, the associated Axiom form document will automatically open as a task pane in the Axiom Assistant area. This approach is recommended when the purpose of the task pane is to perform actions on that particular spreadsheet file.
- You can embed the Axiom form within a custom task pane, using the command **Show Form**. When using this approach, the task pane can be opened in any way that a normal custom task pane can be opened. For example, the task pane can be assigned to open automatically at startup, or it can be opened on demand from the Task Panes Library. This approach is recommended when the purpose of the task pane is more general, and does not depend on the presence of any particular file.

NOTE: Regardless of which configuration you use, any user whom you want to use the task pane must have at least read-only security permission to the Axiom form. There is no configuration that implicitly grants access to the Axiom form when it is used as a task pane. Even if the form is embedded in a custom task pane and the custom task pane is specified as a startup file, the user still needs security access to the Axiom form (though they do not need access to the task pane file).

Comparison of using Axiom forms versus regular custom task panes

The primary reason to use an Axiom form over a regular custom task pane is the ability to gather inputs from the user and then pass those values from the Axiom form to the active spreadsheet file (using the "form state" feature). Custom task panes cannot do this.

For example, you could create a form task pane to be used as a persistent refresh assistant for the current file. The user could make selections in the task pane and apply them to the current file using buttons in the task pane.

However, there are other advantages to using an Axiom form as a task pane, even if you do not need to pass values to the active spreadsheet file. For example, the appearance of the form task pane is completely customizable—Axiom forms have much more design flexibility than custom task panes.

On the other hand, Axiom forms do not support the full range of commands that custom task panes support. If you want to use a task pane as an alternative "menu" where users can access various Axiom Software features, then you should use a custom task pane.

Designing an Axiom form to be used as a task pane

The Axiom form to be used as a task pane should be configured as follows:

- If you want the task pane to interact with the active client spreadsheet, then you can configure the file to use form state and/or various commands that execute on the active spreadsheet file.
    - For more information on using form state, see Passing values between an Axiom form and the active client spreadsheet (form state).
    - For more information on available commands, see Executing commands on the active client spreadsheet from an Axiom form.
- The **Canvas Size** of the form (width x height) should be set to an appropriate size for a task pane. A good starting point is 300 x 800. Generally speaking, task panes should be tall and thin. You can use dynamic sizing and positioning options for components as appropriate, so that the form will adjust to the user's task pane area.

  To help you design the form to fit the target canvas size, make sure to enable **Options > Show form canvas area** in the Form Designer.

- If a Formatted Grid component is used, and if the task pane may be associated with multiple files that could be open at the same time, then the number of rows and columns in the grid must remain constant. If the number of rows or columns changes as a user moves from file to file, then the update and refresh behavior of the Axiom form may not work as expected.

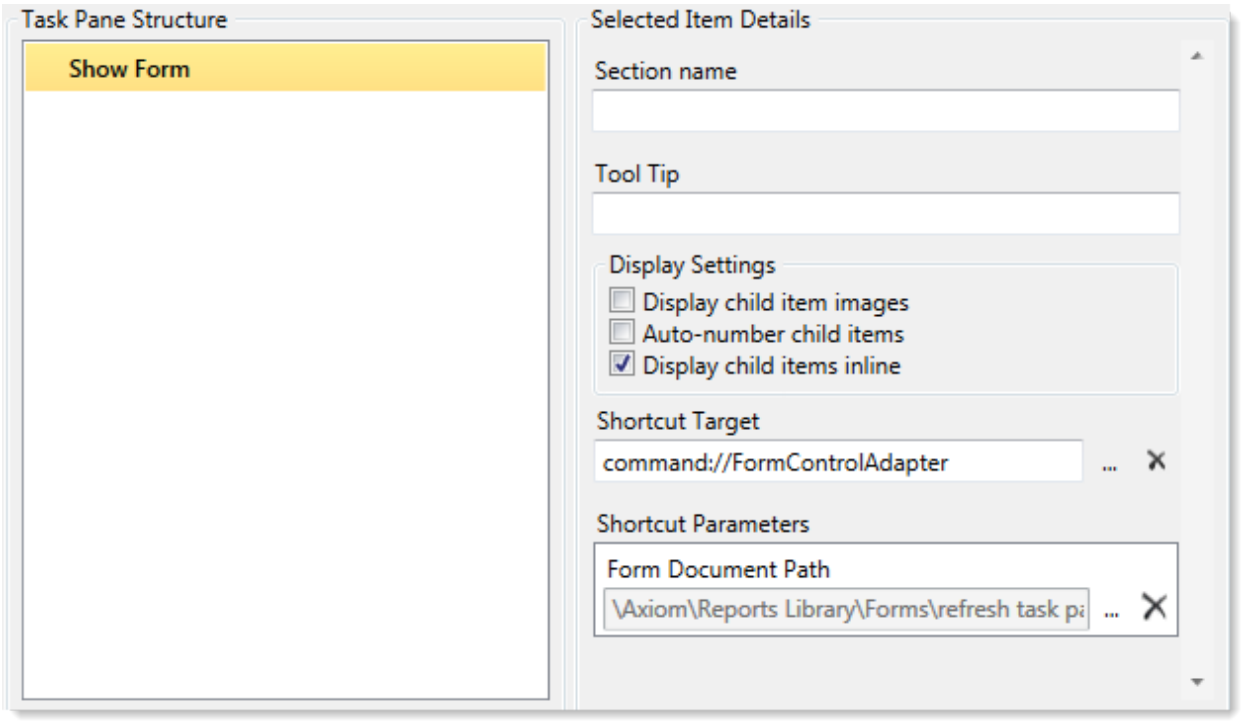
The form size can be set in the Form Properties dialog. To open this dialog, click **Edit Form Properties** at the top of the Form Assistant or the Form Designer.

When deciding where to save the file and what to name it, keep in mind that another file may require the continued presence of this file (if using the **Associated Task Pane** option). If the Axiom form is deleted, moved, or renamed, then the file that is configured to use the Axiom form as an associated task pane may no longer work as expected. You may want to name the file so that the relationship between the two files is clear, or save it to a designated folder that holds required "supporting" files such as this one.

Embedding a form within a custom task pane

Use the **Show Form** command in a custom task pane to embed the contents of an Axiom form within a custom task pane. This allows the Axiom form to be used in any context that a custom task pane can be used—such as a startup file.

It is required to configure the task pane so that the **Show Form** command is the only item in the task pane, and to enable **Display child items inline** for that item. This means that when the task pane is opened, the designated Axiom form is displayed as the entire contents of the task pane.



Recommended configuration for using Show Form

For more information on creating custom task panes and using the Show Form command, see the *System Administration Guide*.

Associating a form task pane with a particular file

To associate a task pane with a particular Axiom file, you must complete the **Associated Task Pane** setting on the Control Sheet of the spreadsheet file. When the Axiom file is opened, the associated task pane is also automatically opened and will remain available for use while the file remains open.

For example, if you want to associate a form task pane with a particular report, you would enter the file path to the form in the Associated Task Pane setting for the report:

```
\Axiom\Reports Library\Forms\MyTaskPane.xlsx
```

When the report is opened, the form-enabled file named `MyTaskPane.xlsx` is opened in the Axiom Assistant area as a task pane, and that task pane remains linked to the report file.

Alternatively, you can use document shortcut syntax to specify the form. When using a document shortcut, you can add a parameter to specify an alternate tab name for the form. If specified, this tab name will be used instead of the file name. For example:

```
document://\Axiom\Reports
Library\Forms\MyTaskPane.xlsx?AxiomTabName=MyTab
```

To create the document shortcut syntax, take the normal file path and then add the text `document://` to the front of it. To use the optional tab name parameter, append the text `?AxiomTabName=`*`Name`* to the end of the shortcut.

For more information on using this feature, see the discussion on task panes in the System Administration Guide.

NOTE: If multiple open files have the same associated task pane, then each open file will have its own instance of the Axiom form. Each instance will remain in sync with its own file.

Configuring close options for a form dialog

When a form is open as a dialog in the Excel Client or Windows Client, you can use the **Close Dialog** command on a Button component, to provide end users with a button or buttons that close the dialog.

NOTE: This command only applies to form dialogs, such as refresh forms or forms that are opened via Show Form Dialog (either the command or the function). If the command is used in any other form context—such as in a form task pane, or in a form opened in the Web Client browser—the command is ignored.

Typically, software dialogs have two buttons that can be used to close the dialog. To illustrate the purpose of these buttons, they will be referred to as the "OK" button and the "Cancel" button (though you can label the buttons using any text).

- **OK button**: This button is used to perform an action and close the dialog. The action could be to save data to the database, or to apply form state values to the active spreadsheet file. In this case, the user has decided that they are finished making choices in the dialog, and now they want those choices to be used and the dialog to be dismissed.
- **Cancel button**: This button is used to close the dialog without performing any action. In this case, the user has decided that they do not want to perform any action, and they simply want the dialog to be dismissed.

You can have variations of these buttons depending on the purpose of the dialog. For example, you could have an "Apply" button that performs the action but leaves the dialog open. And you could decide not to have a Cancel button, and instead require the user to cancel out of the dialog by clicking the **X** button on the title bar.

Configuring Close Dialog for an OK button

When using the Close Dialog command for an OK button, keep in mind the following:

- The processing step for the command must be set to **Active Client Spreadsheet - After Processing**. This means that the full form update process will be performed in addition to closing the dialog, thereby allowing other actions to be executed.
- If the button uses any other commands that will be run at the same processing step, you do not need to worry about the order of these commands. The Close Dialog command is always performed first at this processing step, however, any other commands will still execute after the dialog is closed.

Configuring Close Dialog for a Cancel button

When using the Close Dialog command for a Cancel button, the processing step for the command must be set to **Active Client Spreadsheet - Before Processing**. This means that the dialog will be closed before the normal form update process occurs, thereby aborting the form update process. No actions will occur.

Passing values between an Axiom form and the active client spreadsheet (form state)

When an Axiom form is used as a dialog or a task pane within the Axiom Software Excel Client or Windows Client, it can accept values from the active client spreadsheet and then pass changed values back to that spreadsheet. *Active client spreadsheet* means the spreadsheet that is currently open and active in the client when the form dialog or task pane is opened. This interaction between form and spreadsheet is accomplished by using the "form state" feature for Axiom forms.

When using form state, certain designated values are stored in memory. This "form state memory" can then be shared between the active client spreadsheet and the form (dialog or task pane). When the form is opened, the existing form state values in the active client spreadsheet are passed into the form. Users can then use the form to change the form state values, and then pass the new values back to the active client spreadsheet.

For example, you may have a form state key named Dept, to store a department code. You can set a default value for Dept in the spreadsheet file, and pass it into the form when it is opened. You can use interactive components in the form to change the value of Dept, and pass that changed value back to the spreadsheet file. Both files can reference the Dept value as needed, such as to impact data queries.

Setting form state values

Form state values can be set as follows:

- In the Axiom form, interactive components can be configured to use the FormState tag, in order to store the interactive value in memory. For example, instead of storing the selected value of a combo box to the Selected Value field in the Form Control Sheet, the selected value can be saved to memory using a designated form state key, such as `[FormState=Dept]`.
- In the Axiom form, a FormState data source can be used to set any form state value. This option can be used when the value you want to save is the result of a calculation in the spreadsheet, instead of the direct selection of a interactive component.
- In either file, the GetFormState function can be used to set a default value if the form state key does not yet have a set value. This is the primary means of setting "starting values" in the spreadsheet file, to pass into the form. It can also be used in the form to set starting values.
- In the spreadsheet file, the ShowFormDialog function can be used to set form state values as part of launching the form dialog, thereby causing the values to be passed into the form.

Returning form state values

Form state values can be returned as follows:

- In either file, the GetFormState function can be used to return the value for a designated form state key. For example, `GetFormState("Dept")` returns the current value of Dept.
- In the Axiom form, if an interactive component is configured with a FormState tag, that component will return the current value of the designated form state key. For example, a combo box that is configured with `[FormState=Dept]` will return the current value of Dept in the combo box. However in this case, the interactive component can also be used to change the value, by selecting a new value from the combo box.

Passing form state values

Form state values can be passed between the files as follows:

- When the form is opened, the current form state values for the active client spreadsheet are automatically passed into the form. If the form is being opened as an associated task pane, then there is only one opportunity to pass values from spreadsheet to form, when the task pane is initially opened. But if the form is opened as a refresh form or by using the ShowFormDialog function, then every time the form is opened, the current spreadsheet values are passed into the form.
- In the Axiom form, values can be passed from the form to the active client spreadsheet by using a button configured with the **Apply Form State** command. This command takes the current values in the form and passes them to the spreadsheet, including the ability to trigger a refresh within the spreadsheet.

Form State Example

Imagine that you want to use an Axiom form as a "refresh form" for a report. You want the user to be able to select values within the Axiom form to determine what data will be brought into the report. In order to share the selected values in the Axiom form with the active client spreadsheet, you must save these selected values to form state memory.

For example, the Axiom form contains a combo box where a user can select a VP name. You want to pass this selected VP value to the report file so that it can be used to define a filter for an Axiom query. This process works as follows:

- The **Selected Value** cell for the combo box is configured with a FormState tag, such as `[FormState=VPName]`. This tag tells Axiom Software to store the value in form state memory instead of writing it back to the cell, and define the identifying key name for this particular form state value ("VPName").
- The user selects VP name "Jones" from the combo box. This value is stored in form state for the Axiom form, as the current value for VPName.
- The user clicks a command button that is configured with the **Apply Form State** command. This command sends the current form state values from the Axiom form to the active client spreadsheet (the open report in this case).
- The report file is configured with GetFormState functions to read the form state values. When the form state values are updated from the Axiom form, the function GetFormState ("VPName") returns the value Jones. This value can now be used in the Axiom query filter.

Setting up a form to use form state

When an Axiom form is used as a dialog or task pane in the Desktop Client, it can use form state to pass values between the form and the active client spreadsheet.

To use form state in an Axiom form, you can:

- Set up one or more interactive components with a FormState tag to store that component's value in form state memory. You can also optionally use a FormState data source to store any value in form state memory (meaning values that are not explicitly associated with a form component).
- Set up a Button component in the form with the Apply Form State command. This button is used to pass the values from the form to the active client spreadsheet.
- Use the GetFormState function in the form as needed to set default values, and to return the current form state value in the form.

Using the FormState tag with form components

You can configure an interactive component in an Axiom form to store its value in form state memory instead of within a designated cell in the file. This enables the value to be passed to the active client spreadsheet using form state.

To configure a component to store its interactive value as a shared variable, use the FormState tag. Use of the FormState tag differs slightly depending on what type of interactive component you are configuring:

- For stand-alone components that typically place the interactive value in a component property—such as the Selected Value property for a ComboBox component—the FormState tag is placed in the relevant component property on the Form Control Sheet.
- For content tags that typically place the interactive value in a target cell in the spreadsheet—such as when using the Select tag in a Formatted Grid component—the FormState tag is included as a parameter in the content tag (instead of using the TargetCell parameter).

The syntax for the FormState tag is as follows:

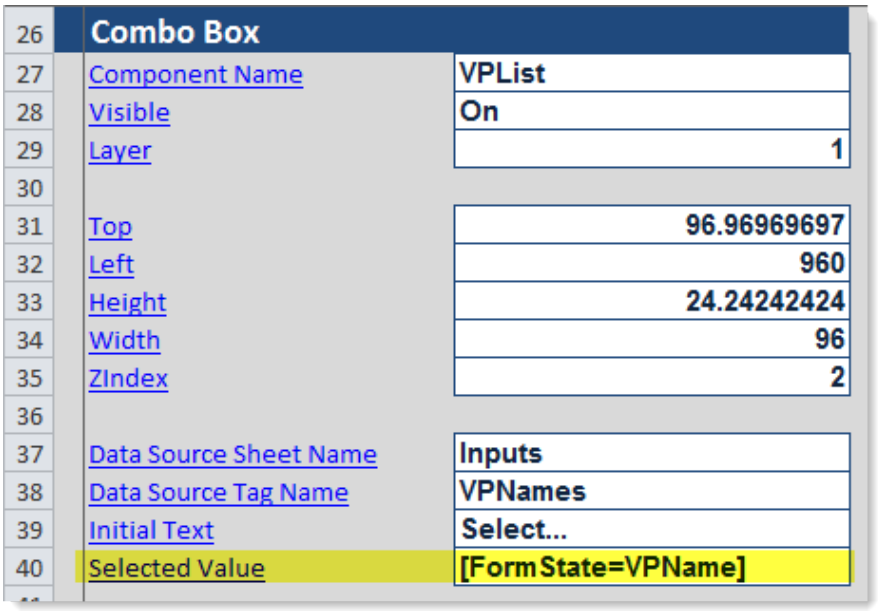
```
[FormState=KeyName]
```

The presence of the FormState tag tells Axiom Software that you want to store the value in form state memory instead of within a designated cell. The KeyName defines the name under which the value will be stored in form state memory.

For example, if you want the value of a Combo Box component to be stored in form state, you would place the following tag in the **Selected Value** cell for the component:

`[FormState=VPName]`

Where the combo box is used to select a VP name. If VP "Jones" is selected from the list, this value will be stored in form state memory as VPName=Jones, and the combo box will display the value Jones.



26	**Combo Box**	
27	Component Name	VPList
28	Visible	On
29	Layer	1
30		
31	Top	96.96969697
32	Left	960
33	Height	24.24242424
34	Width	96
35	ZIndex	2
36		
37	Data Source Sheet Name	Inputs
38	Data Source Tag Name	VPNames
39	Initial Text	Select...
40	Selected Value	[FormState=VPName]

When using the FormState tag, the contents of the Selected Value cell itself are never changed. The selected value is only stored in memory; it does not get written to the file. If you need to reference the selected value elsewhere in the file, use a GetFormState function to return the value. The GetFormState function is also the only way to define a default value for the component within the form.

As noted, FormState tags can also be used with content tags in Formatted Grid components. For example, if you want the value for a Select tag to be stored in form state, you would set up the Select tag as follows:

`[Select;DataSourceName=VPNames;Placeholder=Select a VP;FormState=VPName]`

This Select tag does not contain a TargetCell parameter. Instead, the FormState parameter is used to save the user's selection in form state.

When using the FormState tag, the form state value is only set when a user explicitly interacts with the component and sets a value. Until that occurs, the component will display the current value of the form state key. If a value is passed in from the active client spreadsheet, the component will use that value. Otherwise, the component will display a default value set in the form file by use of the GetFormState function. If the form state key does not have any value, then the component will display no value (which may mean that placeholder text displays instead).

Form state can be used with the following form components:

Component	Feature or Setting	Notes
Check Box	Is Checked	To set a default value using the GetFormState function, use True (checked) or False (unchecked). This is also how the function returns the variable value after a user has interacted with the check box.
Combo Box	Selected Value	N/A
Date Picker	Selected Date	• To set a default value using the GetFormState function, use a date string such as "12/31/2018". • When returning a selected value using the GetFormState function, it will be returned as a date/time value with the time set to midnight, regardless of the cell formatting. If you want to use cell formatting to change the display, wrap the function in a DateValue function, such as: `=DateValue(GetFormState ("SelectedDate"))`
Formatted Grid	CheckBox tag DatePicker tag Select tag Selected Row ID TextArea tag	• To set a default value for the CheckBox tag using the GetFormState function, use 1 (checked) or 0 (unchecked). This is also how the function returns the variable value after a user has interacted with the check box. • To set a default value for the DatePicker tag using the GetFormState function, use a date string such as "12/31/2018". • If the text input for a TextArea tag contains line breaks, then the cell containing the GetFormState function must have Wrap Text enabled in order to display those line breaks when returning the value using the function. • If the TextArea tag is numeric, the number format will be taken from the cell containing the tag. Normally the number format is taken from the target cell, but when using form state there is no target cell. • The notes for the Text Box component also apply when using the TextArea tag.
Hierarchy Chart	Selected Value	N/A
Map View	Selected Value	N/A

Component	Feature or Setting	Notes
Pie Chart	Selected Label	N/A
Text Box	Text	• When returning the variable value using the GetFormState function, it is returned as a string value, regardless of the text box type and the cell formatting. If you want to use cell formatting to display a numeric value, wrap the function in a Value function, such as: `=Value(GetFormState("Total"))` This formula will error if the form state key does not have a value, so you can either use an IF function to handle the no value case, or you can define a default value in the GetFormState function. • If the text box type is Input Mask, and Preserve Input Mask is enabled, then any default value in the GetFormState function must use the input mask format if you want it to display that way. Once the actual value is set, it will automatically use the input mask.
Toggle Switch	Is Checked	To set a default value using the GetFormState function, use True (checked) or False (unchecked). This is also how the function will return the current state after a user has interacted with the check box.

Components can be configured to auto-submit or not as desired. Apply Form State does not execute until the full form update process is complete, so any unsubmitted component values will be sent back to the form source file before the form state values are applied to the active client spreadsheet.

Defining form state values using a FormState data source

Using the FormState data source, you can store any value in form state memory. When using this approach, the form state keys and values are not associated with any particular component—you can define any value and pass it to the active client spreadsheet using form state.

For example, you may want to take the selected value of an interactive component and perform some logic on it using formulas, and then send the resulting value to the active client spreadsheet. This is only possible using the FormState data source.

The tags for the FormState data source are as follows:

Tag Type	Tag Syntax
Primary tag	`[FormState;DataSourceName]` The DataSourceName for FormState data sources is not currently referenced anywhere, but it may be used in the future. Data source names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save. The placement of this tag defines the control column and the control row for the data source. • All column tags must be placed in this row, to the right of the tag. • All row tags must be placed in this column, below the tag.
Row tags	`[FormStateItem]` Each row that defines a form state value must be flagged with this tag.

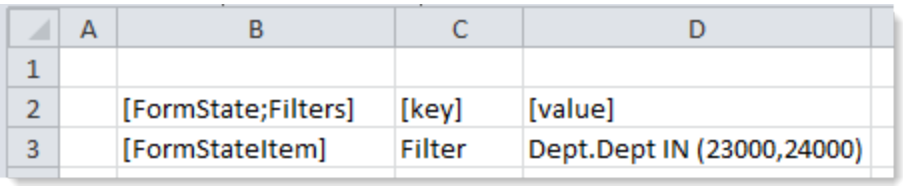
Column tags	`[Key]`
	Defines the key name for the form state value. This is the name that will be used in the GetFormState function to return the value.
	`[Value]`
	Defines the value for the key name.

NOTES:

- The primary tag must be placed in the first 500 rows of the sheet.
- Formulas can be used to create the tags, as long as the initial bracket and identifying keyword are whole within the formula.

You can manually type the FormState data source tags in the spreadsheet, or you can use the Data Source Wizard to add them. To use the wizard, right-click a cell and select **Create Axiom Form Data Source > Form State**. You can add the tags first and then complete the necessary values, or you can define the values first and then highlight them and right-click to add the tags.

The following screenshot shows a sample FormState data source:



	A	B	C	D
1				
2		[FormState;Filters]	[key]	[value]
3		[FormStateItem]	Filter	Dept.Dept IN (23000,24000)

In this example, the value is set by a formula that reads the various selected values of a MultiSelect data source, and creates a filter criteria statement based on those values.

When you use a FormState data source, the form state values are set at the end of the update cycle, right before the form web page is refreshed. This means that if you are using any formulas to determine the form state values, they will reference values that have completed the full form update cycle—including updating component values and refreshing Axiom queries.

One way in which the FormState data source differs from using the FormState tag is that the FormState data source is executed whenever the form is updated, including the update that occurs when the form is initially opened. This means that the form state keys in the data source are always set to the values in the data source when the form is opened.

Because of this behavior, any form state values that are passed in from the active client spreadsheet when the form is opened will be immediately overwritten by the execution of the FormState data source. If you want to use the value passed in from the active client spreadsheet, then you must use a formula similar to the following in the `[Value]` cell:

```
=IF(F15="",GetFormState("MyKey","MyDefault"),F15)
```

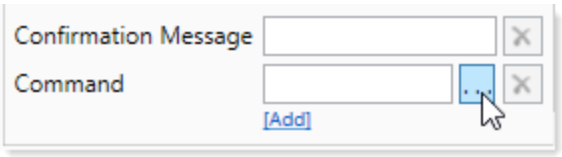
This means if cell F15 is blank, then use the result of the GetFormState function to set the value. The GetFormState function will return the value passed in from the active client spreadsheet, or if there is no value, it will set a default value. Otherwise, use the value in F15. Cell F15 would then contain whatever formulas you want to use to set the form state value within the form itself.

Setting up a Button component to Apply Form State

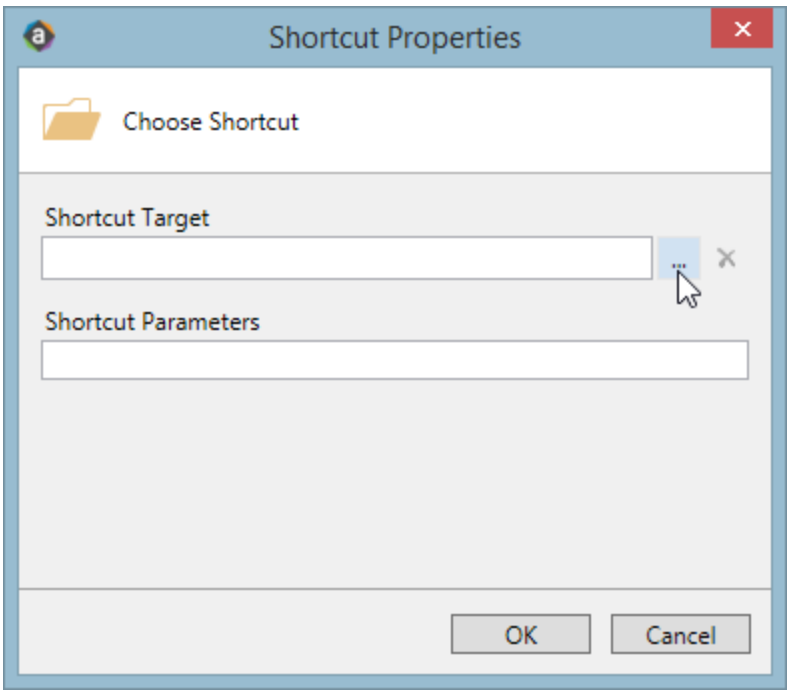
To pass form state values from the Axiom form to the active client spreadsheet, you must use a Button component that is configured with the **Apply Form State** command. When the user clicks the button, the form state values in the Axiom form are passed to the active client spreadsheet. Additionally, the spreadsheet file is refreshed according to the refresh mode specified in the shortcut properties for the command.

To start off, add the Button component to the Axiom form canvas and then configure the properties as desired. The button text should be defined as something like "OK" or "Apply" (depending on how the Axiom form will be used). You can then configure the **Command** for the component as follows:

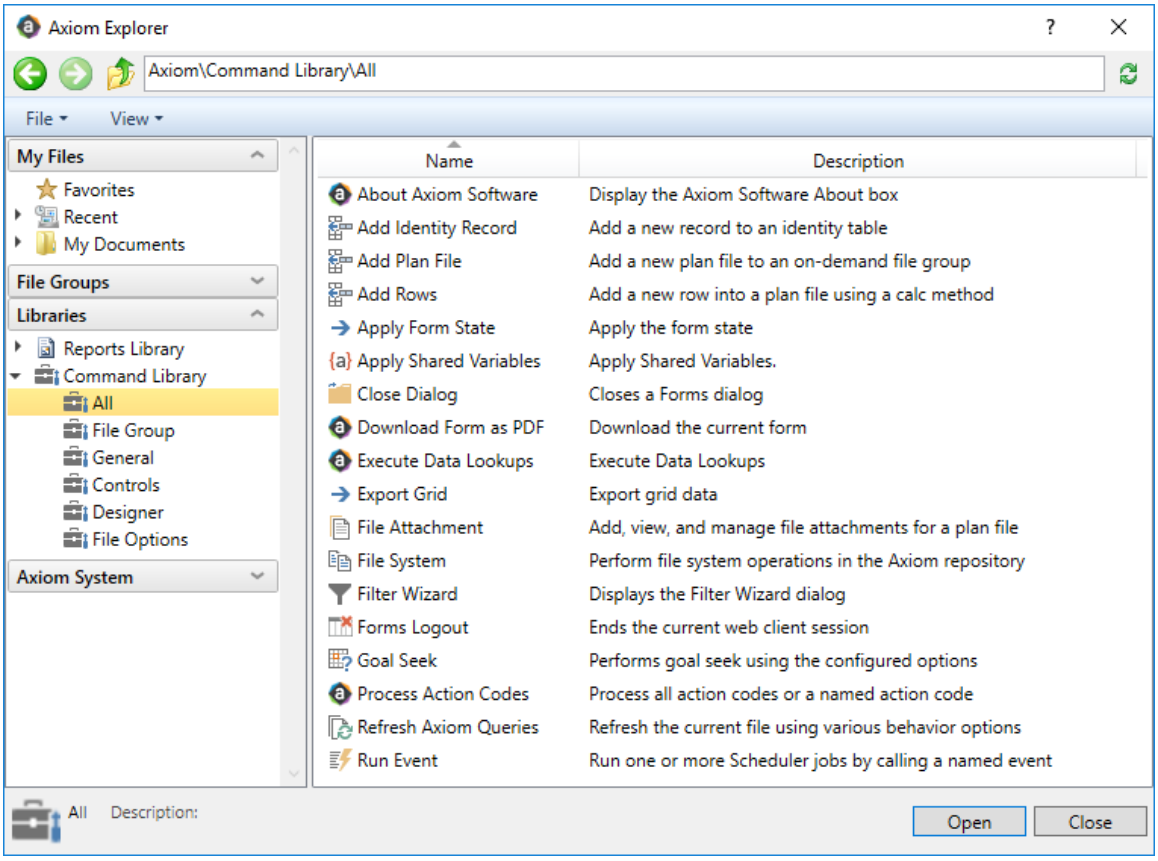
1. In the Button component properties, click the **[...]** button to the right of the **Command** box.



2. In the **Shortcut Properties** dialog, click the **[...]** button to the right of the **Shortcut Target** box.



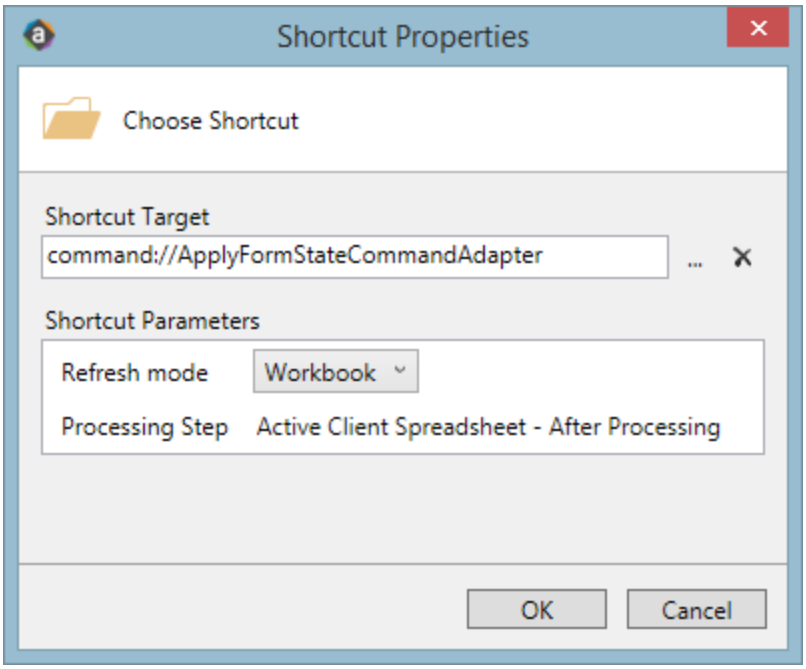
3. In the **Axiom Explorer** dialog, navigate to the **Command Library**. Select the **Apply Form State** command, then click **Open**.



Example Command Library for Axiom forms

The Apply Form State command is now listed as the target in the Shortcut Properties dialog, and the relevant shortcut parameters are now available.

4. In the shortcut parameters, select the desired **Refresh mode**, then click **OK** to close the Shortcut Properties dialog.



Example Shortcut Properties dialog

The refresh mode determines the level of refresh that occurs in the active client spreadsheet after applying the form state values. The refresh mode can be one of the following:

- **Calculate only** (default): Formulas are recalculated for the workbook. Axiom queries are not refreshed.
- **Worksheet**: Formulas are recalculated and Axiom queries are refreshed for the current worksheet only.
- **Workbook**: Formulas are recalculated and Axiom queries are refreshed for all sheets in the workbook.
- **None**: No calculation or refresh occurs.

NOTE: The refresh mode for the command is ignored if the Axiom form is used as a refresh form. In this case, the command honors the refresh context that caused the Axiom form to display as a refresh form. For example, if the user selected to refresh the active sheet only, then the command will refresh the active sheet only, regardless of the refresh mode setting.

The **Processing Step** for Apply Form State is fixed and cannot be changed. It is always executed on the active client spreadsheet, after the full form update process has completed. For more information, see Axiom form update process.

Once the form state values have been passed from the Axiom form dialog to the active client spreadsheet, those values can be read within that file using the GetFormState function.

NOTE: When using Apply Form State in a refresh form or in a form dialog, you should also add a Close Dialog command to the same button. This means that when a user clicks the button, the form state values will be applied and then the dialog will be automatically closed. In most cases this is the desired behavior for a modal dialog. For more information, see Configuring close options for a form dialog. However, Close Dialog does not apply to form task panes and should not be used in that context. Form task panes are intended to be persistent and remain open while the associated spreadsheet file is open.

Using the GetFormState function in the form

The GetFormState function can be used to return the current value of a given form state key, or to return a default value if no value has yet been set. This function can be used in both the Axiom form and in the active client spreadsheet as needed.

The GetFormState function takes the following parameters:

`GetFormState("KeyName","DefaultValue")`

Where *KeyName* is a form state key name. The DefaultValue parameter returns a default value if the actual value for the key has not yet been set.

The main reason to use the GetFormState function in the form source file is to return the current value of a form state key, so that it can be used within the form itself. For example, imagine that you have one component where a user selects a grouping column (such as Dept, VP, or Region), and another component where the user selects specific items in that column. You want to be able to dynamically point the second component to the correct column. If the first component stores its value in the form state key Grouping, then you can construct the appropriate table column using a formula like the following:

`="Dept."&GetFormState("Grouping")`

For example, if the current value of Grouping is VP, then this formula returns `Dept.VP`.

You can also use the GetFormState function to define a default value within the form source file. This default value may be useful while you are designing and testing the form. However, keep in mind that if a default value is specified in the active client spreadsheet as well, this value will override the default value set in the form source file. For more information, see Understanding how form state values are passed between workbooks.

Using form state values in the active client spreadsheet

The active client spreadsheet is the spreadsheet that is active in the Desktop Client when the Axiom form is opened as a dialog or a task pane. You can set up this spreadsheet to return the form state values that are passed from the Axiom form and use these values in some way, such as to filter the data in an Axiom query. You can also set default form state values in the spreadsheet that will be passed to the Axiom form when it is opened.

Returning form state values

When the **Apply Form State** button is used in the Axiom form, the current form state values in the Axiom form are passed to the active client spreadsheet. In order to use those values within the spreadsheet, you must use the GetFormState function to return them.

The GetFormState function takes the following parameters:

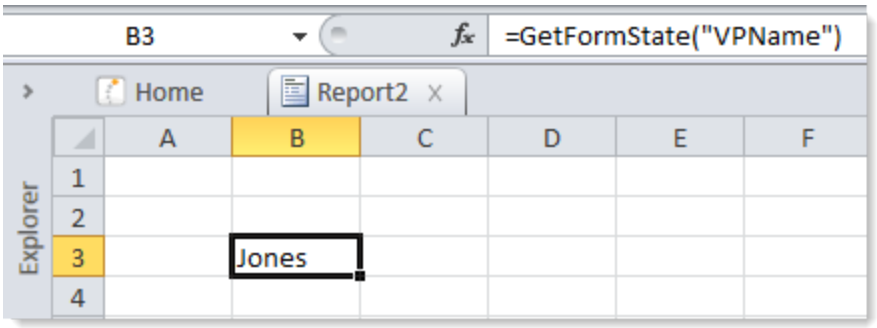
`GetFormState("KeyName","DefaultValue")`

Where *KeyName* is a form state key name. The DefaultValue parameter returns a default value if the actual value for the key has not yet been set.

For example, imagine that the Axiom form contains a Combo Box component that is set up with the following FormState tag: `[FormState=VPName]`. When a user selects a value from the combo box—for example, Jones—that value is stored in form state as the VPName value. In order to read this value within the active client spreadsheet, you would use the following formula:

`=GetFormState("VPName")`

This returns Jones when Jones is the currently selected value for the combo box (the value stored in form state memory).



When using the GetFormState function in the client spreadsheet, keep in mind the following:

- The GetFormState function returns form state values for *the current file*, it does not read the values from other files. When GetFormState is used in the client spreadsheet, the function does not know what the form state values are in the Axiom form until those values are passed from the form to the spreadsheet (thereby becoming the form state values for the spreadsheet).
- Form state values can only be set within an Axiom form (or by launching an Axiom form using ShowFormDialog). Until the form state values are set, the GetFormState functions in the client spreadsheet will return blank unless the DefaultValue parameter of the function is used to set a default value for each form state key.

The GetFormState function simply returns the current form state value; it does not have any other impact on the spreadsheet file. It is up to the file designer to use that value in a way that impacts the spreadsheet—for example, to use the value in an Axiom query filter, or to include the value in a save-to-database process.

Setting form state values

In the active client spreadsheet, you can set a default value for a form state key within the GetFormState function. For example:

```
=GetFormState("VPName","Smith")
```

This example defines the default value of VPName as "Smith." This function will return "Smith" if VPName does not yet have a set value. Once VPName has a set value, then the default value is ignored and the GetFormState function simply returns the actual form state value.

Defining a default value in the function does not actually set the value for the form state key. You can change the default value in the function, and as long as no "real" value has yet been set for the form state key, the function will update to return the new default value. However, once a form state value has been set by a form (using Apply Form State), then the function will return that value and the default value is ignored.

In this example, the GetFormState function in the spreadsheet returns the default value of Smith. When the Axiom form is opened, this default value is passed into the form and becomes the current form state value for VPName in the form (overriding any default value set in the form). If an interactive component such as a combo box is configured to set the value for VPName, then you can use the component to change the value to something like Jones. This sets a "real" value for the VPName form state key. When the Axiom form passes its form state values to the client spreadsheet using Apply Form State, the GetFormState function in the spreadsheet will now return the value Jones. Now that a real value has been set for the form state key in the form, the default value in the function is ignored.

Generally speaking, form state values are set in the form, not in the active client spreadsheet. The active client spreadsheet can only set default values. However, there is one exception to this rule. When using the ShowFormDialog function to open an Axiom form dialog, you can define form state values in the function and set those values in both the spreadsheet and the form when the function is used. For example:

```
=ShowFormDialog("View Details","\Axiom\File Groups\Budget
2019\Utilities\Details.xlsx";"Acct=6000")
```

When a user double-clicks on the function to open the designated form dialog, the value 6000 is set for the form state key Acct. This is a "real" form state value that is set in the spreadsheet file and then passed to the form when it is opened, overriding any current value for Acct in both files. Depending on the purpose of the form, the form may simply use this value (such as to impact Axiom queries in the form), or the form may provide a way to further change the value and send the new value back to the spreadsheet using Apply Form State.

Understanding how form state values are passed between workbooks

When an Axiom form is opened as a dialog or a task pane in the Desktop Client, any form state values in the active client spreadsheet are automatically passed to the Axiom form. It does not matter whether the spreadsheet file and the Axiom form have a defined relationship (such as the form being assigned as the

file's refresh form or as an associated task pane) or whether the spreadsheet file just happens to be active when the Axiom form is opened.

This behavior occurs when the form is launched using any of the following methods:

- Using the ShowFormDialog function in the active client spreadsheet, or using the ShowFormDialog command in a custom task pane or ribbon tab
- Using **Refresh** to open a form designated as a refresh form for the active client spreadsheet (the `[RefreshDialog]` tag)
- Using the **Associated Task Pane** property to open an Axiom form as the associated task pane for the active client spreadsheet

When the Axiom form is opened, the starting value for a form state key is determined as follows:

- If the active client spreadsheet has a value for a particular form state key, that value is passed to the Axiom form and used as its initial form state value.
- If the active client spreadsheet does not have a value for a particular form state key, then any default value defined in the Axiom form by the GetFormState function is used.
- If no value is set in either location, then the form state key will not have an initial value when the Axiom form is opened.

After the Axiom form has been opened and the initial form state values are set, the form state values can be changed as follows:

- For interactive components that are configured to use form state, the form state values are set when the interactive component is changed. This happens at the start of the form update process.
- For FormState data sources, the form state values are set as part of the form update process, near the end of the process (just before the After Saving Data processing step).

Finally, when the user clicks the **Apply Form State** button in the form, the full form update process occurs and then the current form state values are sent from the form to the active client spreadsheet.

This behavior allows for a "round trip" of form state values from the form, to the spreadsheet, and then back to the form, as illustrated in the following example.

Example

A report file is set up to use an Axiom form as a refresh form. The Axiom form and the report contain GetFormState functions as follows:

Axiom form

```
=GetFormState("Region","North America")
```

Report

```
=GetFormState("Region","Europe")
```

When the report is first opened, the GetFormState function returns the default value of Europe.

When the report is refreshed and the Axiom form is opened, the form state value of Europe is sent to the Axiom form and used as the value for Region. This is because the report sends its current form state values to the Axiom form when the form is opened, and "overrides" any default values set in the Axiom form itself. If the report did not have a value defined for Region, then the Axiom form would use the default of North America as defined in the form.

Now imagine that the user selects Asia as the Region value in the form, and then uses Apply Form State to pass that value to the report. The GetFormState function in the report now returns the value Asia. When the report is refreshed again and the Axiom form is opened again (within the same file session), the value of Asia is now sent to the form and the form uses that value for Region.

If the report is closed, its form state values are cleared. So the next time the report is opened, it is back at the starting point where it uses the default value of Europe as set in the GetFormState function.

This behavior also illustrates why it is important to define unique form state key names, so that you do not accidentally send values between files that should not be sharing values. For example, if you reuse the key name "Year" in many different contexts, you may run the risk of accidentally setting a year value from an entirely different context. Using more specific key names, such as "CapitalRequestYear," helps to protect against accidental form state collisions.

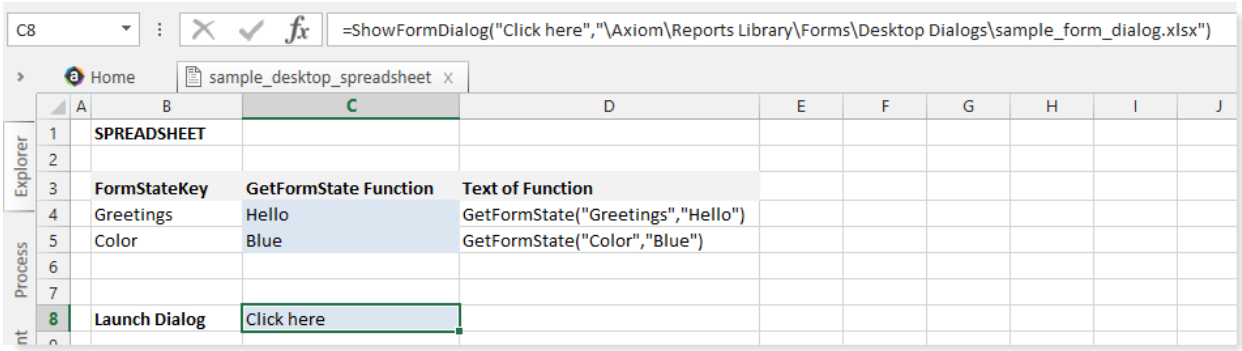
You should also take care not to place multiple GetFormState functions in the same file with different default values for the same key name. There should only be one instance of the GetFormState function in a file for each unique key name. You can place these functions on a separate Variables sheet and then use cell references in other sheets to reference the form state values.

Form state example

The following example illustrates how form state values can be set and passed between the active client spreadsheet and an Axiom form dialog or task pane. This is not a "real life" example; the intent is simply to show the basic concepts involved.

File setup

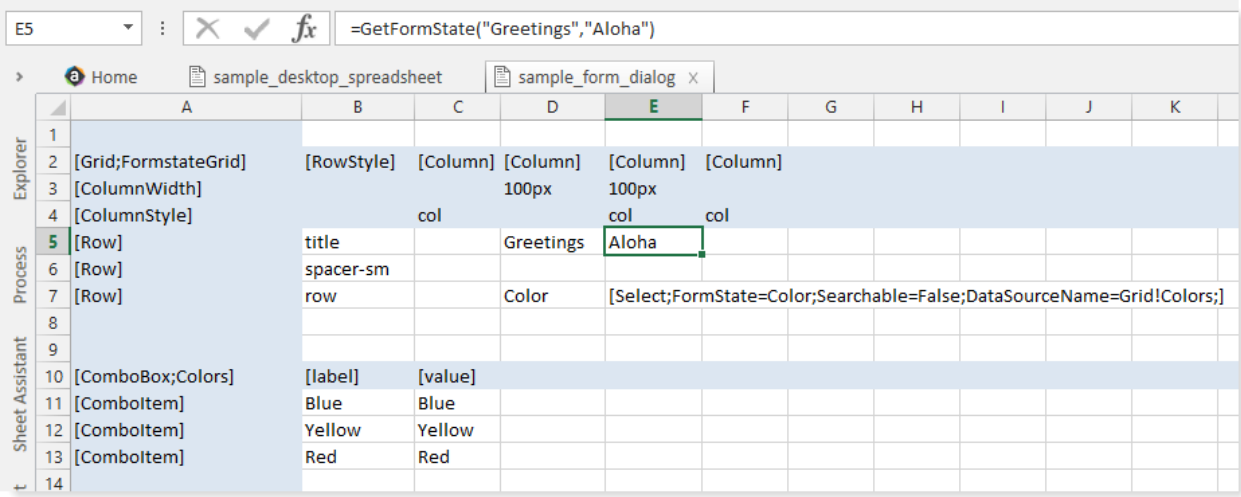
The active client spreadsheet contains GetFormState functions to define default form state values and to return values set by the Axiom form.



In this example, there are two form state keys, Greetings and Color. Cells C4 and C5 contain the actual GetFormState functions and define default values of "Hello" and "Blue" respectively. The actual text of the GetFormState functions are shown in column D for reference.

Lastly, cell C8 contains a ShowFormDialog function, which will be used to launch the Axiom form as a dialog. Although this is the approach that this example uses to launch the Axiom form, the same behavior would apply if the form was launched as a task pane (as an associated task pane for the file) or if the form was launched as a refresh form (using the `[RefreshDialog]` tag).

The Axiom form to be launched is set up as follows:

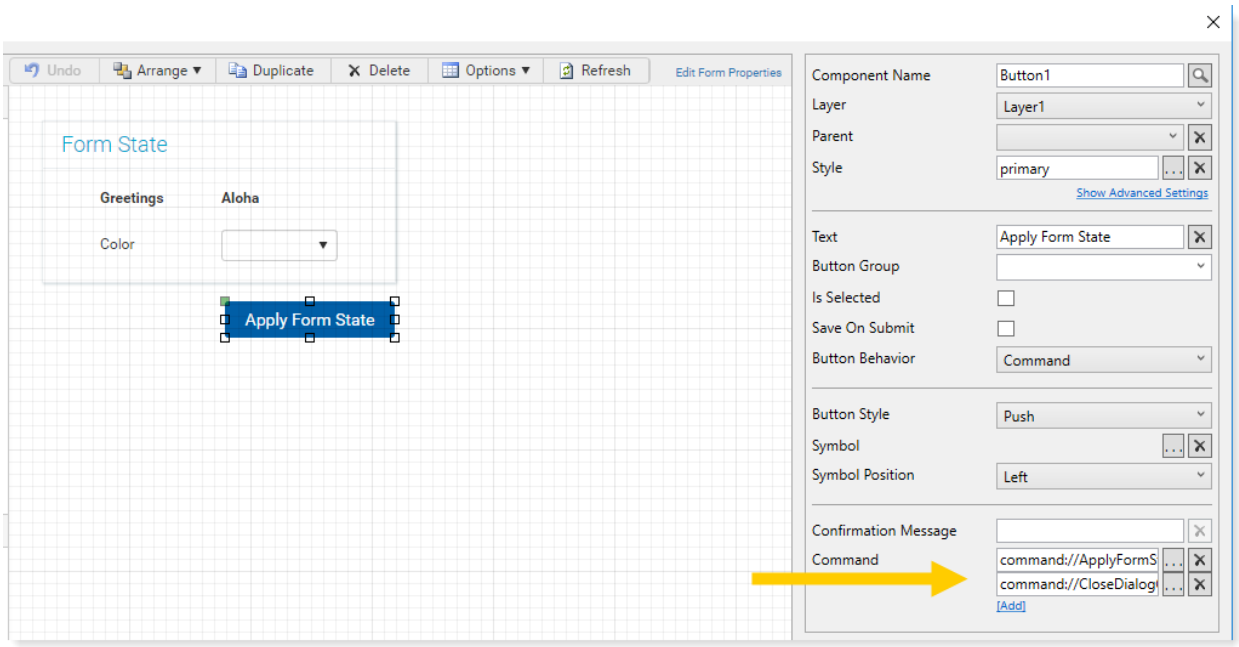


The form state key of Greetings will be returned into cell E5 using a GetFormState function. You can see that the formula here uses a different default value of "Aloha". The form state key of Color is used by a Select tag, as defined in cell E7. You can see that the Select tag is configured to store its value into `FormState=Color` instead of in a target cell. This means that the combo box for the Select tag will

return the current value of Color, and it can be used to change the current value of Color. The data source for the Select tag is the ComboBox data source shown in the screenshot.

In this example, we are using a Formatted Grid component to display the form state values and to change the Color value (using the Select tag). We could have done the same thing by using a separate Label component and a ComboBox component, but it is easier to show the setup within a single grid. Also note that the form is only set up to read the Greetings value, not to change it. This is done to show both potential ways of using a form state value in an Axiom form. If desired, we could have set up the Greetings value to be changed as well (such as by using a TextArea tag).

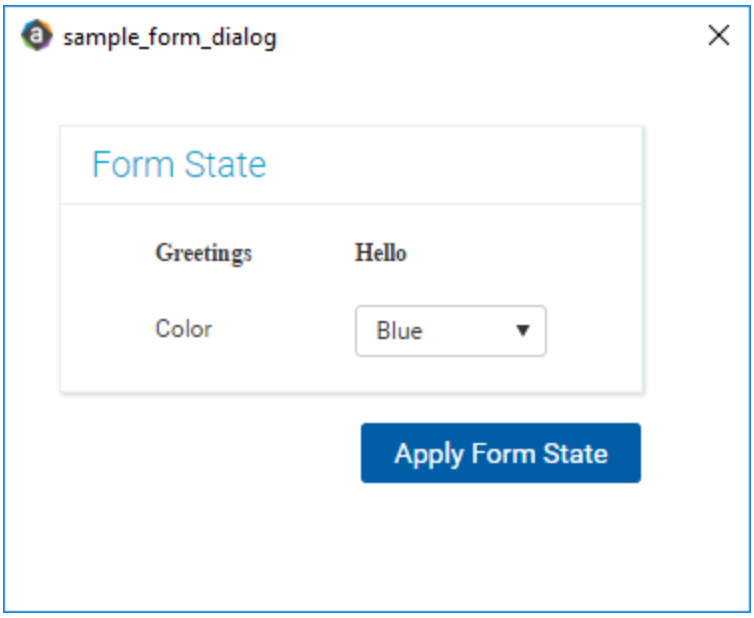
In order to pass changed form state values back to the active client spreadsheet, the form must be set up with a Button component that uses the Apply Form State command. The following screenshot shows this button in the Form Designer:



The button has two commands, Apply Form State and Close Dialog. Apply Form State is used to pass the current form state values to the active client spreadsheet, and to refresh the spreadsheet. The Close Dialog command closes the dialog when the process is complete, so that the user is now back within the active client spreadsheet. In this example, the button is named Apply Form State for clarity, but in real form dialogs it is more likely to be named something like OK or Apply.

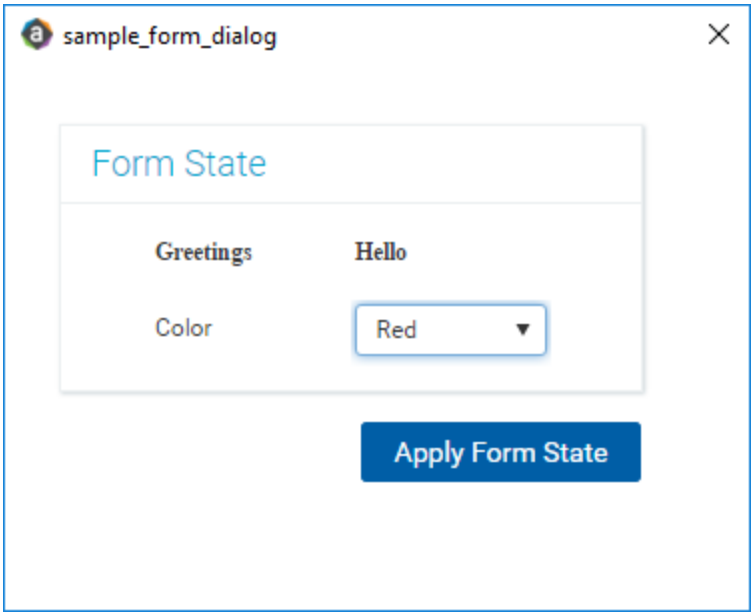
Passing and changing values in the files

When a user double-clicks the ShowFormDialog function in the spreadsheet file, the designated Axiom form opens. As part of this process, the current form state values in the spreadsheet are passed to the form, overwriting any default values set in the form. So when the form is opened, it appears as follows:

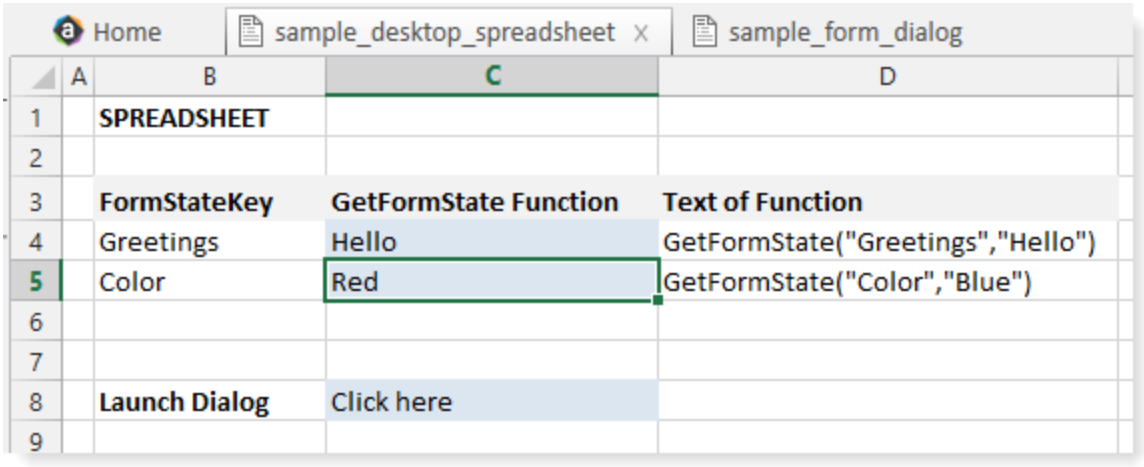


In this dialog, you can see that the "Hello" default value for Greetings from the spreadsheet has overwritten the "Aloha" default value in the form. If the spreadsheet did not have a defined value for Greetings, then the form would have started with Aloha instead. Similarly, the combo box for the Select tag currently displays the "Blue" default value for Color that was passed in from the spreadsheet.

Now the user can use the form to change form state values by using interactive components. In this example, the user interacts with the combo box to change the Color value from Blue to Red.



When the user is done changing values in the form, they click the Apply Form State button. This passes the form state values to the active client spreadsheet, and refreshes the spreadsheet. In the spreadsheet, the GetFormState functions now look as follows:



	A	B	C	D
1		**SPREADSHEET**		
2				
3		**FormStateKey**	**GetFormState Function**	**Text of Function**
4		Greetings	Hello	GetFormState("Greetings","Hello")
5		Color	Red	GetFormState("Color","Blue")
6				
7				
8		**Launch Dialog**	Click here	
9				

You can see that the GetFormState function for Color now returns Red, which is the value that was set in the Axiom form and passed to the spreadsheet. The spreadsheet's default value of Blue is now ignored. Because the Greetings value was not changed by the Axiom form, the GetFormState function for Greetings continues to display Hello.

If the user were to launch the form dialog from the spreadsheet again, now the current value of Red would be passed into the Axiom form, and the combo box would start with the value of Red. The user could change the value in the form again, and pass it back to the spreadsheet again.

If the spreadsheet is closed and reopened, the form state key of Color would revert back to the default of Blue. Whenever a file is closed, its form state memory is cleared.

This is a simple example, intended to illustrate the basic flow of form state values between the form and the active client spreadsheet. In real life, the form state keys and the passed values are likely to be values that impact data. The form and/or the spreadsheet would be configured to use the values in some way, such as to filter the results of an Axiom query.

Executing commands on the active client spreadsheet from an Axiom form

When an Axiom form is used as a dialog within the Axiom Software Excel Client or Windows Client, it can execute commands on the active client spreadsheet by use of a Button component. *Active client spreadsheet* means the spreadsheet that is currently open and active in the client when the form dialog is opened.

For example, you can use the Process Action Codes command on a Button component in a form to execute action codes on the active client spreadsheet instead of in the form's source file (which is the

default context for the command). The *processing context* for the command indicates where the command will be executed.

For purposes of this discussion, the term "dialog" refers to any use of Axiom forms within a controlled user interface. This includes using an Axiom form as a refresh form, a task pane, or as a generic dialog.

Supported commands

Only certain commands are eligible to be executed on the active client spreadsheet from an Axiom form.

Command	Description	Notes
Apply Form State	Apply form state values to the active client spreadsheet	• This command is only for use in Axiom form dialogs that interact with the active client spreadsheet. It has no use in standard forms. • The command can optionally trigger a refresh in the active client spreadsheet after form state values are applied. However, if the form dialog is used as a refresh form, the refresh always occurs when Apply Form State is used.
Close Dialog	Close the form dialog	• This command is only for use in Axiom form dialogs. It has no use in standard forms. • This command does not apply to form task panes. These associated task panes open and close with their associated file; they cannot be closed separately.
Goal Seek	Perform goal seek calculations	• This command can be performed on either the form source file or on the active client spreadsheet. The processing context for the command determines where it is executed. • When configuring the shortcut parameters for the command, the context of any cell references depend on the processing context. If processed on the form, then the cell references apply to the form source file. If processed on the active client spreadsheet, then the cell references apply to that file.

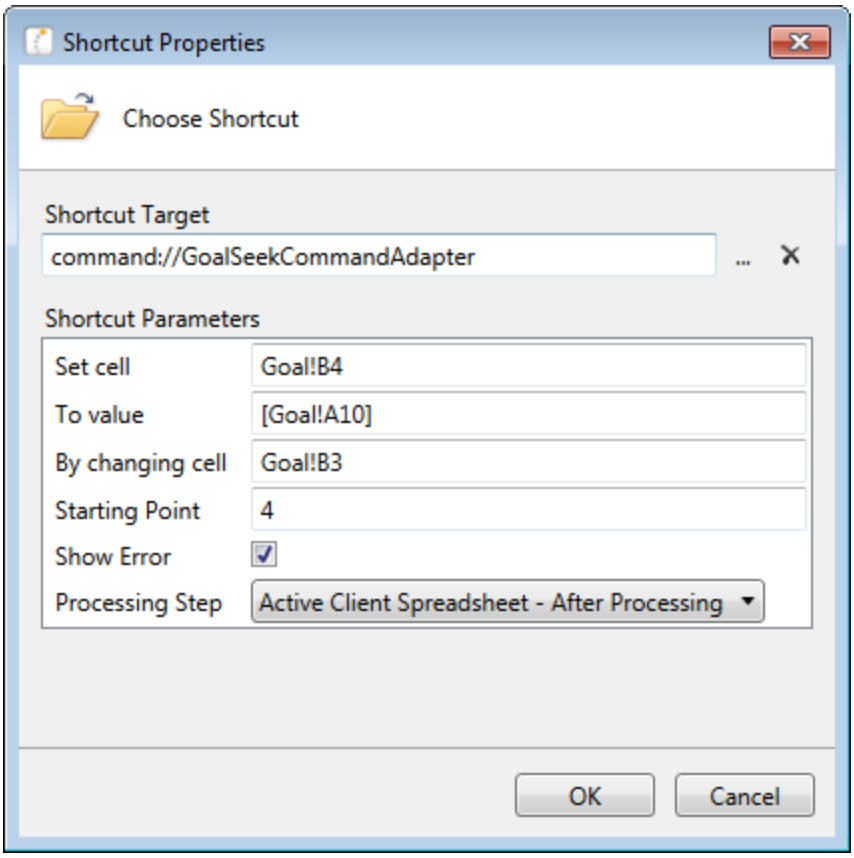
Command	Description	Notes
Process Action Codes	Process action codes to copy or lock cells	• This command can be performed on either the form source file or on the active client spreadsheet. The processing context for the command determines where it is executed. • When configuring the shortcut parameters for the command, the designated sheet and action code names depend on the processing context. If processed on the form, then these settings apply to the form source file. If processed on the active client spreadsheet, then these settings apply to that file.
Refresh Axiom Queries	Trigger a refresh using various refresh options	• This command is only for use in Axiom form dialogs that interact with the active client spreadsheet. It has no use in standard forms (because the normal form update process refreshes Axiom queries).

Commands can be executed by using either a Button component or a Button tag in a thematic Formatted Grid. However, in many cases you will want to combine commands on a single button so that you can execute the action and close the form dialog in a single button click. Because Button tags only support a single command, the Button component is more likely to be used for this particular use case.

Configuring a command to execute on the active client spreadsheet

The **Processing Step** for the command determines both *where* and *when* the command will be executed. This is configured in the shortcut properties for the command. The first part of the option indicates where the command will be executed, and the second part of the option indicates when the command will be executed. For example, if the processing step is **Active Client Spreadsheet - After Processing**, then this means the command will be executed on the active client spreadsheet, after the full form update process is complete.

In order to execute a command on the active client spreadsheet, the processing step must be prefixed by **Active Client Spreadsheet**. You can then choose to execute the command either Before Processing or After Processing. For some commands, the processing step is not configurable, and displays for information only.



Example command configured to execute on the active client spreadsheet

For more information on processing steps, see Timing of command execution and Axiom form update process.

Commands and form state

Usually, any use of commands on the active client spreadsheet will be performed in conjunction with sending form state values. This is because the typical purpose of the form dialog is to guide the user in making certain choices and/or inputs, and then these values are sent to the active client spreadsheet to affect the other commands being performed on the spreadsheet.

However, it is also possible to execute commands on the active client spreadsheet without any use of form state. For example, you could have a form task pane with a button that refreshes Axiom queries on the active client spreadsheet, without also sending any form state values. In this case, the form task pane would simply be presenting an alternate user interface for triggering a refresh.

Reference

This section contains reference information for use in Axiom forms.

Axiom Form Components

Axiom forms are created by placing various components on the form canvas, and then configuring the data and display properties of those components. The following components are available to be used in Axiom forms.

Form Controls

- Button: Users can click a button to refresh the Axiom form (including a save-to-database if applicable), and to perform a configured action.
- Check Box: Users can select or clear a check box. The state of the check box is submitted back to the source file.
- Combo Box: Users can select an item from a list. The selected item is submitted back to the source file.
- Data Grid: Query data from the Axiom Software database and display it in a grid.
- Date Picker: Users can select a date from a calendar. The selected date is submitted back to the source file.
- Formatted Grid: Display information in a formatted spreadsheet-style grid. Users can edit unprotected cells in the grid, and can select rows in the grid. The edited cell contents and / or selected row are submitted back to the source file.
- Hyperlink: Users can click the hyperlink text to open a web page or a document.
- Image: Display an image such as a company logo.
- Label: Display small amounts of user-defined text, such as for titles, descriptions, or contact information.
- Radio Button: Users can click one of a set of radio buttons to select an option for the Axiom form. The selected button is submitted back to the source file.
- Slider: Users can slide a button along a predefined range to specify a value. The selected value is submitted back to the source file.

- Text Box: Users can type text into the text box. The text is submitted back to the source file.
- Toggle Switch: Users can toggle the switch to Off or On. The state of the switch is submitted back to the source file.

Feature Controls

- Announcements: Users can view and manage announcements.
- Dialog Panel: Places a set of pre-configured components on the form canvas, to be used as a dialog that users can open from within the form.
- Embedded Form: Display another form embedded within the current form.
- Form Help: Users can click a help icon to open a panel that displays custom help content for the current form.
- Menu: Users can select items from a menu to change the content shown in the current form, or to open web content in a new window.
- Process Summary: Users can view new and important process tasks, and access tools to manage these tasks.
- Titled Panel: Places a set of pre-configured components on the form canvas, to be used as a template for further design of a titled form.
- Wizard Panel: Users can move through a defined set of steps and complete a configured action.

Charts

- Area Chart: Display data in an area chart.
- Bar Chart: Display data in a horizontal bar chart.
- Bubble Chart: Display multidimensional data in a bubble chart.
- Bullet Chart: Display a current value and a target value along a defined measurement scale.
- Column Chart: Display data in a vertical column chart.
- Hierarchy Chart: Display data in an expandable and collapsible hierarchy.
- KPI Panel: Display one or more KPI values.
- Line Chart: Display data in a line chart.
- Linear Gauge: Display a value along a defined measurement scale, with the scale presented in a linear format.
- Map View: Display geospatial data on a map view.
- Pie Chart: Display data using a pie chart.
- Scatter Chart: Display multidimensional data in a scatter point chart.
- Scatter Line Chart: Display multidimensional data in a scatter line chart.
- Sparkline Chart: Display trend data in a simple at-a-glance chart.
- Radial Gauge: Display a value along a defined measurement scale, with the scale presented in a radial format.
- Waterfall Chart: Display data in a waterfall chart.

Shapes

- Ellipse: Draw a circle or ellipse.
- Horizontal Elbow Line: Draw a horizontal line or arrow with elbow bends at each end.
- Rectangle: Draw a square or rectangle.
- Straight Line: Draw a straight line or arrow.
- Vertical Elbow Line: Draw a vertical line or arrow with elbow bends at each end.

NOTE: The components in the Charts and Shapes sections are only available for use if your license provides full access to Axiom forms. For more information, see Licensing requirements for Axiom forms.

Component properties can be configured using the Form Assistant task pane or the Form Designer unless otherwise noted. All properties can also be defined on the Form Control Sheet directly if desired. For example, if you want a property to be dynamic depending on the result of a formula, you can define that formula in the control sheet. To access the control sheet settings for the component, double-click any property name to go to that property in the Form Control Sheet.

Color styles

The following color style codes can be used in Axiom forms and web reports, in areas such as:

- Row and column styles for Formatted Grid components (Axiom forms)
- Label component styles (Axiom forms)
- Icon colors for Data Grid components (Axiom forms)
- Background colors for KPI Panel components (Axiom forms and web reports)

When used in an Axiom form, the skin must be set to Axiom2018 in order to recognize the color style.

Common component properties

Some component properties are common to all components.

General properties

The following general properties are available for all components:

Item	Description
Component Name	The name of the component. This is for identification in the file; this name does not display on the Axiom form canvas. The name of the component identifies the corresponding settings for the component on the Form Control Sheet. The component names are also useful if you have multiple types of the same component within an Axiom form, so that you can tell which component you are currently editing. Component names must be unique within a file and must start with a letter. Names can only contain letters, numbers, and underscores. Names are validated when the file is saved; an invalid name will prevent the save. **NOTE:** Spaces are not allowed in component names and will be automatically removed by Axiom Software. For example, if you enter "My Component" as the component name, it will be automatically adjusted to "MyComponent".

Item	Description
Visible	Specifies whether the component is visible on the Axiom form (On/Off). By default this is set to On. This setting can be used to dynamically show or hide the component using a formula. Keep in mind that if you have multiple components that you need to dynamically show or hide based on the same condition, then it is preferable to place those components on a dedicated layer and then show or hide the entire layer instead of the individual components. **NOTE:** This setting is only available on the Form Control Sheet; it cannot be set in the Form Assistant or in the Form Designer.
Layer	The layer that the component belongs to on the Axiom form canvas. In the Form Assistant and the Form Designer, this displays as the layer name (for example: Layer 1). In the Form Control Sheet, this is recorded as the layer ID (for example: 1). If the canvas only has one layer, then the component is automatically assigned to that layer and cannot be changed. If the canvas has multiple layers, you can assign the component to any layer using the drop-down list. By default, the component will be assigned to whichever layer is selected in the Layers box when you initially drag the component onto the canvas. For more information on layers, see Using multiple layers on the canvas. If desired, you can jump to the applicable layer settings on the Form Control Sheet by clicking the binoculars icon next to the drop-down list.
Parent	The parent component that this component is assigned to. If blank, then the component does not have an assigned parent. Currently, only Panel components can be designated as parents. If a component has an assigned parent, then that component is positioned within the parent instead of within the canvas at large. If the parent is hidden, all "child" components of that parent are also hidden. The parent assignment is automatically completed when a component is dragged into a panel in the Form Designer, and automatically cleared when a component is dragged out of a panel. In most cases, you should not need to manually assign a parent. For more information, see Using panels to group and position components.

Style and formatting properties

To define the component formatting, you can assign one or more styles to the component. Styles can impact formatting properties such as fonts, borders, and colors.

If you do not want to apply a style to this component, or if you want to override one or more formatting properties in an assigned style, click the **Show Advanced Settings** link underneath the **Style** box to display the individual formatting properties. For more information on defining individual formatting properties for a component, see Formatting overrides for Axiom form components.

Item	Description
Style	Optional. The styles used to determine the formatting of the component. You can assign one or more styles. Click the **Select component styles** button **[...]** to open the **Choose Style** dialog. Using this dialog, you can select one or more styles to apply to the component. The available styles depend on the component type and the skin assigned to the form. For more information, see Using component styles. Some components have several styles that are specifically designed for that component type, while other components may only have the "generic" styles that are available to all components. When using a generic style, keep in mind that they may not be useful for all components. You can view a description of each style and view the effective formatting applied by the selected styles within the Choose Style dialog.
Component Theme	(Deprecated.) The theme to use for the component instead of the form-level theme. If left blank, the component uses the form-level theme. This setting should be left blank unless you need to override the form theme. Generally speaking, themes should be set at the form level and only overridden at the component level when necessary. This setting is available in the advanced component properties (click **Show Advanced Settings** under the **Style** box). On the Form Control Sheet, the setting displays using the name **Theme Override**. **NOTE:** This setting only applies if your form uses a legacy skin (any skin except the default Axiom2018). The Axiom2018 skin does not use themes.

Position and size properties

You can view the position and size properties for a component by clicking the **Show Advanced Settings** link under the **Style** box. If necessary, you can edit these properties directly (instead of automatically modifying them by adjusting the component's position and size on the canvas). For more information on using these settings, see Controlling component position and size.

Item	Description
Reference Location	The reference location determines how the x-position and y-position of a component are evaluated. By default the reference location is UpperLeft. **NOTE:** This setting is not exposed in the advanced component settings. It can be changed on the canvas by double-clicking the corner selection handles of a component, or you can edit the setting on the Form Control Sheet directly.
X Position Y Position	The x-position determines the component's position along the horizontal axis, and the y-position determines the component's position along the vertical axis. Both are evaluated relative to the reference location. Positions can be set in pixels (default) or percentages.
Width Height	The width and height determine the size of the component. The width and height can be set in pixels (default) or percentages. Size keywords are also available to support special behavior.
Rendering Order	The order in which the component is rendered in the layer. A component with a larger order number will display above a component with a smaller order number. For components that support tab navigation (tabbing to the next editable component), the rendering order also determines the tabbing order. **NOTE:** On the Form Control Sheet, this setting is labeled as **Z-Index**.
Lock Layout	If enabled, the component size and position are locked and cannot be changed by dragging and dropping on the canvas. This optional setting is intended to protect against accidentally moving or resizing a component while working on the canvas.

Formatting overrides for Axiom form components

The formatting applied to each Axiom form component is primarily determined by its assigned style or styles. Styles can define formatting properties such as text color, font size, border color, background color, and so on. Some formatting properties may be defined in the style itself, and other formatting properties may be inherited from the skin. For more information, see Controlling the Axiom form appearance with skins and styles.

If a component does not have an assigned style, or if you want to override a specific formatting property of the assigned style, then you can use the advanced component settings to define certain formatting properties. If a formatting property is defined for a component using these advanced settings, it will take precedence over any equivalent properties defined in the style or skin.

To access the advanced component properties:

- In the Form Designer or Form Assistant, click the **Show Advanced Settings** link under the **Style** box.

This exposes the individual formatting properties that are available for the current component. You can now edit any of these properties as desired. When you are finished, click **Hide Advanced Settings** to restore the default view and hide these properties.

NOTE: The advanced settings also include showing the position and size properties for the component. For more information on these properties, see Controlling component position and size.

Component formatting properties

The following table details the formatting properties that may be available for each component in the advanced settings. Some properties do not apply to certain components or may not be configurable at the component level. If a formatting property does not display for a particular component type, then it is not available for that type.

Each formatting property can be defined individually and only affects that particular property. Any formatting properties left blank in the advanced settings will continue to use the formatting defined in the style or skin. If you have previously defined a formatting property and now you want to clear it to return to using the inherited property, you can click the **Clear selection** button to the right of the property.

Item	Description
Text Color	The text color. If left blank, the color is determined by the style or skin (in that order). Click the **[...]** button to open the **Choose Color** dialog. You can select from the colors displayed at the top of the dialog, or you can enter a valid RGB or hexadecimal color code (such as #00FFFF for Aqua). Click **OK** to use the specified color. If you are modifying the Form Control Sheet directly, then you must use a hexadecimal code. For an example list of colors and hexadecimal codes, see: http://www.w3.org/TR/css3-color/#svg-color. **NOTE:** For the Text Box component, the text color does not apply to any text defined in the Placeholder field (except in Internet Explorer). It will apply when text is defined for the Text field.
Font Size	The font size of the text, in pixels. If left blank, the size is determined by the style or skin (in that order).

Item	Description
Font Family	The font type to use for the text. If left blank, the font family is determined by the style or skin (in that order). If you specify a font family, it is strongly recommended to use a common font such as Arial, which all client machines and devices are likely to support. If the specified font is not found on the user machine, then it will be ignored and the next font in the formatting order of precedence will be used.
Font Weight	The weight of the font. You can specify any of the following: • <Blank>: The font weight is determined by the style or skin (in that order). • **Normal**: The font weight is normal. • **Bold**: The font weight is bold. • Any valid CSS entry for font weight, such as `500`. The drop-down list contains entries for 100-900.
Font Style	The style of the font. You can specify any of the following: • <Blank>: The font style is determined by the style or skin (in that order). • **Normal**: The font style is normal. • **Italic**: The font style is italic.
Background Color	The background color for the component. If left blank, the background color is determined by the style or skin (in that order). If the component is inheriting a background color from the style or skin and you want the component to use no background color, then you can specify `transparent` as the color. You must manually edit the Form Control Sheet to do this, because the Background Color selector in the Form Designer / Form Assistant does not allow this entry. The background color can be specified as described previously for Text Color. **NOTES:** • For Panel components, if you want to set the background color you must disable the title bar. • For Wizard Panel components, the background color fills the area underneath the header area and the status bar.
Border Color	The border color for the component. If left blank, the color is determined by the style or skin (in that order). If the component is inheriting a border from the style or skin, and you want to hide the border at the component level, use the **Border Thickness** setting to turn off the border. The border color can be specified as described previously for Text Color.

Item	Description
Border Thickness	The thickness of the border, in pixels. If left blank, the thickness is determined by the style or skin (in that order). If the component is inheriting a border from the style or skin, and you want to hide the border at the component level, you can specify 0. Zero thickness results in no border. **NOTE:** If you are using the border settings for the component, then **Show Title Bar** should be disabled (for components that support this option). If both settings are used, then two borders will display on the component.
Transparency	The transparency of the component, as a percent. If left blank, the transparency behavior is determined by the style or skin (in that order). Currently, Label components are the only components that support setting the transparency at the component level. By default this is set to 0, which means the component is opaque. At the other end of the spectrum, if transparency is set to 100, then the component will be completely transparent (but still present in the form; it is not the same effect as setting Visible to Off). You can set the transparency to a percentage between 0 and 100 to allow the contents underneath the label to show through the label. When any value is specified for the transparency, it is applied to the entire label component (both the text and the background color, if defined). **NOTE:** For legacy skins with white backgrounds (such as Uniform), if the transparency is blank and no background color is specified, then the background is transparent. But if any value is specified for the transparency and no background color is specified, then the background is white. This is to support backward-compatibility with previous versions.
Horizontal Alignment	The horizontal alignment of the text. If left blank, the alignment is determined by the style or skin (in that order). Specify **Left**, **Center**, or **Right**.
Vertical Alignment	The vertical alignment of the text. If left blank, the alignment is determined by the style or skin (in that order). Specify **Top**, **Center**, or **Bottom**.

NOTE: For the Date Picker component, all font-related properties apply to both the date input box and the calendar control. For example, if you use bold font, then the selected date will display in bold, and all of the text in the calendar control (month / day labels, dates, etc.) will also display in bold.

Form Control Sheet

The Form Control Sheet defines settings to render an Axiom form. All general form settings and individual component settings are stored on this sheet.

The Form Control Sheet must be added to a file in order to create an Axiom form, however, once the sheet is added you can perform the setup using the Form Assistant task pane. As you define settings within the Form Assistant and the Form Designer, these settings will be automatically reflected on the control sheet. In general, the only reason to interact directly with the Form Control Sheet is if you need to configure a dynamic formula for a particular setting.

To add a Form Control Sheet to an Axiom file:

- On the **Axiom** tab, in the **File Options** group, click **Forms > Enable Forms for the active document**.

  OR

- On the **Axiom Designer** tab, in the **Developer** group, click **Tools > Add a Control Sheet > Form**. This option is only available to administrators.

The added sheet is named **Control_Form**. The Form Control Sheet is only visible to administrators and to users with Sheet Assistant permission. Otherwise, the sheet is hidden by default.

NOTE: When you first open a form-enabled file after upgrading Axiom Software to a new version, the Form Control Sheet is automatically upgraded for new features. This may result in being prompted to save the file when closing even though you have not made any changes to it. If so, you can save the file now, or if not it will be upgraded again the next time it is opened.

General Form Settings

The top section of the control sheet contains general settings for the Axiom form.

Item	Description	More Information
Title	Optional. If desired, you can define an alternate form title that will display in the browser tab when the form is opened in a browser. This title does not apply when the form is opened as a web tab within the Windows Client. If left blank, the tab text is the file name by default for reports. If the file is part of a file group, the Tab Prefix and Tab Column settings for the file group are honored to determine the tab text.	N/A

Item	Description	More Information
Theme	(Deprecated.) The theme applied to the Axiom form. The theme defines the formatting for the components in the form, by defining certain default formats and determining which styles are available for use in components. By default, the theme is blank, which means default formatting is applied to components and default styles are available for use. Generally speaking, themes are intended to fit certain form use cases, such as the Report theme for forms that display reporting data. If your form fits the use case of one of the available themes, it is recommended to apply that theme. **NOTE:** Themes only apply if your form uses a legacy skin (any skin except the default Axiom2018 skin).	Setting the theme for an Axiom form (deprecated)
Skin	The skin applied to the Axiom form. The skin influences the overall look & feel of the form, by defining certain top-level design elements and default colors. When you create a new form, the skin is automatically set to the system default skin. You can change the skin on a per form basis if needed. The system default skin is controlled by the **WebClientSkin** system configuration setting, which is set to Axiom2018 by default.	Setting the skin for an Axiom form
Width Height	The width and height of the form canvas, in pixels. By default these settings are blank, which means the form is inheriting the canvas width and height from the skin. All skins delivered with the Axiom Software platform use a width and height of 400 x 400. You can override this width and height as needed on a per form basis.	Defining the canvas size of an Axiom form
HelpCode	Optional. A help code to associate with the form, in order to display form-specific help.	Associating an Axiom form with a help code
Scale To Fit	Specifies whether the Axiom form is scaled to fit within the current window. This setting is disabled by default, and is primarily intended to support backward-compatibility only. It is not recommended to enable this setting otherwise. **NOTE:** Line components are not compatible with Scale to Fit.	Defining the canvas size of an Axiom form

Item	Description	More Information
Use Web Client Container	Specifies whether the Web Client Container is enabled for the form: • If enabled (default), then when the form is viewed in the Web Client or the iPad app, it is presented within a standard "container" that provides users with access to features such as navigation links and the Filters panel. • If disabled, then the container does not display regardless of where the form is viewed.	Using the Web Client Container with Axiom forms
Show Save Data Confirmation	Specifies whether a confirmation dialog displays after a save-to-database occurs. • If disabled (default), then no dialog displays to the user after a successful save-to-database occurs. The yellow status bar in the lower left corner still indicates when a save is in process. • If enabled, then a confirmation dialog displays after a successful save-to-database. This dialog simply informs the user that the save completed; it does not display any details about the save. The user must dismiss the dialog to return to the form. An error dialog always displays if the save-to-database experiences errors, regardless of this setting.	Saving data from an Axiom form
Background Color	Optional. The background color for the form, in hexadecimal code. If specified, this color overrides the default background color determined by the form-level skin.	Setting the background color or image for an Axiom form
Background Image Path	Optional. The background image for the form.	Setting the background color or image for an Axiom form

Item	Description	More Information
Background Image Repeat	Specifies the repeat behavior for the background image (if defined): • **Both** (default): The image repeats in a tiled pattern both horizontally and vertically, covering the entire form background. • **Horizontal**: The image repeats in a tiled pattern horizontally, starting at the top left corner of the form and extending all the way across. • **Vertical**: The image repeats in a tiled pattern vertically, starting at the top left corner of the form and extending all the way down. • **None**: The image is not repeated. The image displays in the top left corner of the form.	Setting the background color or image for an Axiom form
PDF Size	The default paper size for the PDF, such as A4, Letter, or Legal. The default size is **Letter**.	Configuring an Axiom form for printing
PDF Orientation	The default orientation for the PDF: **Auto**, **Portrait**, or **Landscape**. The default orientation is **Auto**, which means that the orientation will be determined based on the width to height ratio of the form (for example, forms that are more wide than tall will be set to landscape).	Configuring an Axiom form for printing
Triggering Component	When a component triggers an update to the Axiom form, the name of the triggering component is written to this cell. An update can be triggered by one of the following: • The user makes a change to an interactive component that is configured to **Auto Submit**. • The user clicks a Button component. You can reference this cell elsewhere in the file, for example, to only run an Axiom query when a particular component changes. You could use a formula in the standard Control Sheet so that the query is only active when the Triggering Component cell equals a particular component name.	Referencing the triggering component of an Axiom form update

Item	Description	More Information
Is PDF	Indicates whether a PDF copy is being generated for the form. This setting is system-controlled and will always read **Off** when working in the source file. When a user generates a PDF copy of the form, Axiom will automatically change this value to **On** in the copy, and then refresh the form before creating the PDF. This allows the form to dynamically change for the "print copy" (the PDF), by referencing this cell in formulas. For example, you may want to hide a certain component in the print copy.	Configuring an Axiom form for printing
Is Excel Export	Indicates whether a formatted grid is being prepared for export to Excel. This setting is system-controlled and will always read **Off** when working in the source file. When a user exports a formatted grid to Excel, Axiom will automatically change this value to **On** and then refresh the form before exporting the grid. This allows the grid to dynamically change for the export, by referencing this cell in formulas. For example, you may want to hide certain rows or columns in the grid when exporting. Once the grid has been exported, this value is automatically changed back to Off.	Exporting Formatted Grid contents to a spreadsheet

Component Settings

Each component that you add to the form canvas has a corresponding section on the control sheet. These sections are dynamically added by Axiom Software, and will also be dynamically removed if you delete a component.

The best way to find the properties for a particular component is to use the Form Assistant task pane:

- First, select the component in the canvas thumbnail, which causes the component properties to display at the bottom of the task pane.
- Double-click the name of any component property, such as **Component Name**. This will take you to the exact section of the Form Control Sheet that contains the settings for the selected component, with your cursor in the property that you double-clicked.

For more information on the properties available for each component, see Axiom Form Components. The relevant properties are detailed in each individual component topic.

Layer Settings

Each layer defined for the Axiom form canvas has a corresponding section on the control sheet. These sections are dynamically added by Axiom Software, and will also be dynamically removed if you delete a component.

Layers are listed in the Components section of the control sheet.

Item	Description
Component Name	The name of the layer.
Visible	Specifies whether a layer is visible in the rendered Axiom form. By default, all layers are visible. In order to allow users to hide and show layers, you will need to make this setting dynamic—for example, use a formula that points to the Selected Value for a combo box. For more information, see Configuring layer visibility in the Axiom form.
Is Visible in Designer	Specifies whether a layer is visible in the Form Designer. This is controlled while working in the editor itself, by selecting or clearing the check box for the layer. This setting simply reflects the current status of the layer check box. For more information, see Working with layers in the Form Designer.
Z-Index	The depth of the layer on the canvas, in relation to other layers. A layer with a higher number will display above a layer with a lower number. This setting is adjusted automatically when using the **Layers** section in the Form Designer to set layer order. For more information, see Working with layers in the Form Designer.

Other standard component settings display for layers—such as Parent, Style, and position / size settings—but these settings do not apply to layers.

Form Assistant

The Form Assistant task pane is available to help you design Axiom forms. The Form Assistant displays automatically when an Axiom file is form-enabled (meaning the file has a Form Control Sheet).

NOTE: The Form Assistant is only available to administrators and to users with Sheet Assistant permission for the file.

Axiom form canvas

The top portion of the Form Assistant displays a "thumbnail" version of the canvas, and provides access to several form settings.

Unlike the Form Designer (which displays the canvas in actual size), in the Form Assistant the canvas is zoomed out so that more of the form contents can fit within the thumbnail dimensions. You can adjust the thumbnail width by changing the task pane width, and you can adjust the thumbnail height by moving the "splitter" between the thumbnail and the component properties up or down. The size of the canvas for purposes of this zoom is determined by the fixed-position contents of the form (with the

defined canvas size serving as the minimum size). This zoom is limited to 40%; if all of the contents of the form will not fit into this zoom level, then the thumbnail will have scroll bars.

The following options are available in this section:

Item	Description
Refresh	Refresh the form canvas.
Edit Form Properties	Open the Form Properties dialog to define general form properties such as the skin and the canvas size.
Layers	Specify the layers to show on the canvas. You can click the link to show the list of layers, and then select or clear the check box for each layer as appropriate. This is provided as a convenience so that you do not have to open the Form Designer in order to change which layers display in the Form Assistant. Any changes made here are also reflected in the Form Designer. Remember that this setting does not determine which layers display when the Axiom form is viewed; it only determines which layers show on the canvas for editing.
Copy Link	Generate a URL that can be used to view the Axiom form in a browser. The URL is copied to your clipboard; you can then paste it into an email, document, web page, etc.
Preview	Preview the Axiom form. • When using the Windows Client, the form opens within Axiom Software in a web tab. • When using the Excel Client, the form opens in your browser. Additional preview options are available on the **Forms** menu, in the **File Options** group of the Axiom tab.
Edit Layout	Open the Form Designer to add, remove, or edit components on the form canvas.

Component properties

The bottom of the task pane displays editable properties for the currently selected component in the form canvas. The available properties depend on the component type. For more information on each individual component, see Axiom Form Components.

TIP: If components are stacked in the canvas, a special click action is available to select components that are underneath the current component. Click on the top component to select it, then click again to select the component underneath it, and so on. You can also use the Search button next to the **Component Name** property to search for any component on the canvas.

When you edit a component property in the task pane, the corresponding setting on the Form Control Sheet is automatically updated (and vice versa). In most cases, it is easiest to edit properties within the task pane (or in the Form Designer), but if desired you can edit the settings directly on the control sheet. Editing the Form Control Sheet directly is required if you want to use a formula for a particular property; it is not possible to enter a formula into the Form Assistant or Form Designer.

To go to the section on the Form Control Sheet for a particular component, select the component in the canvas thumbnail, and then double-click any property name to be taken to that property. If nothing happens when you double-click a property, this means the property has not yet been added to the Form Control Sheet. To automatically add it to the control sheet, make an edit to the property using the Form Assistant. You can now navigate to the property and edit it as desired.

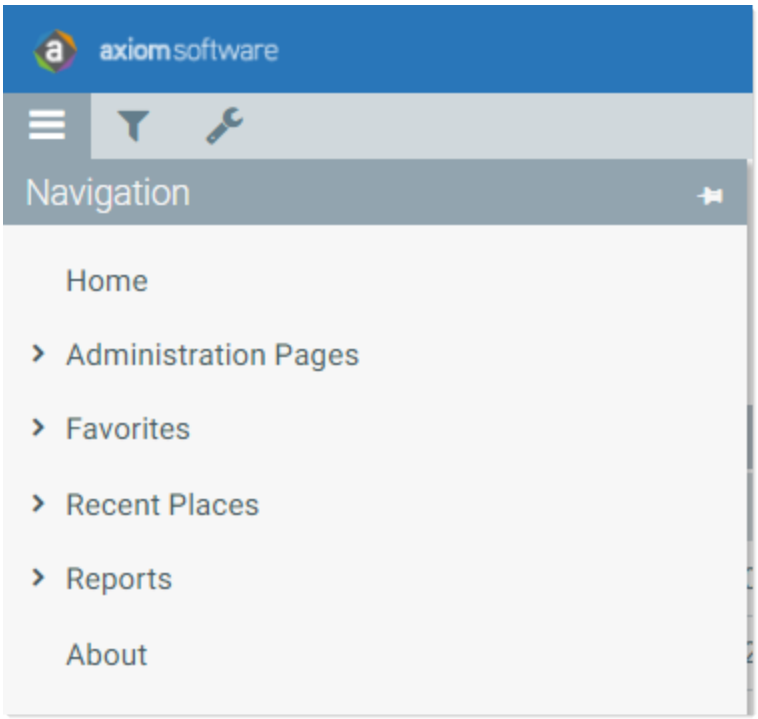
Defining navigation links for the Web Client Navigation panel

You can define a set of navigation links for the Web Client, to provide users with easy access to their Axiom forms, web reports, and various areas of the Web Client. These links are displayed in the left-hand side of the Web Client task bar, in the **Navigation** panel.

You can define two types of navigation links:

- Global navigation links that always display in the Navigation panel, regardless of the currently active page or document.
- Form-specific navigation links that only display when that form is open in the Web Client.

In the Web Client, users can click the menu icon in the top left-hand corner of the task bar to open the Navigation panel and use the links. The global links display at the top of the panel, followed by any form-specific links. Clicking a link opens the designated web page or file within the current window, replacing the currently open form.



The Navigation panel honors the user's security permissions, and hides any links that the user does not have access to.

Web navigation links are defined by using task pane files. Within the task pane files, you can set up links to various web-enabled files as well as to certain areas of the Web Client. When the Navigation panel is rendered, the links are read from the designated task pane files and displayed in the panel as hyperlinks.

The Navigation panel also always contains a link to the Axiom Software launch page, at the bottom of the panel. This link is built-in and cannot be removed or customized.

Defining global navigation links

To define global navigation links, use the reserved task pane file `WebClientNavigationPane.axl`. Only administrators can access and edit this file. This file is located in the Axiom System area of Axiom Explorer:

```
\Axiom\Axiom System\Forms Runtime\WebClientNavigationPane.axl
```

NOTE: Do not change the name of this file. The navigation feature looks for this specific file name.

This file serves as a starter template, containing several sample sections and links. You can use the file as is, or modify the file as needed to meet the needs of your installation. For more information on how to define the navigation links within the task pane document, see Using the task pane editor to define web navigation links.

The Forms Runtime folder also contains a copy of the platform template used to create this file. The name of the platform template is `Platform.WebClientNavigationPane`. This file is updated every time you upgrade your Axiom Software database, so that you always have access to the current platform template. At any time, you can make a copy of this template to "start over" with a new WebClientNavigationPane file.

The starter template contains the following sections and links by default:

- **Home**: Links to current user's Home file. The Home file must be forms-enabled, or else the link navigates to the built-in Forms browse page.
- **Administration Pages**: Links to Table Manager page and the Admin Tools page ("Admin Home").
- **Favorites**: Links to current user's web-enabled favorites.
- **Recent Places**: Links to current user's recent files and places visited in the Web Client.
- **Reports**: Links to current user's web-enabled reports (Axiom forms or web reports), grouped by folders in the Reports Library. Only folders with web-enabled reports will display.
- **About**: Opens Web Client About box.

NOTE: There is no way to disable the global navigation links. If the file is deleted or renamed, the menu icon will continue to display in the Web Client Task Bar, but the navigation panel will be blank (unless there are form-specific links to display).

Defining form-specific navigation links

Form-specific navigation links use the Associated Task Pane feature. To set up form-specific links:

- Create a task pane file in which to define the links. You can name this file anything you like, and save it anywhere in the Task Panes Library. For more information on how to define the navigation links within the task pane file, see Using the task pane editor to define web navigation links.
- In the form where you want the links to display, specify the task pane file as the **Associated Task Pane** for the form source file. This setting is located on the default Control Sheet, in the **Workbook Options** section.

When the form is opened in the Web Client or the iPad app, the links will be read from the designated Associated Task Pane file, and appended to the Navigation panel (below the global navigation links). If the user navigates to a different form, the form-specific links will be removed from the Navigation panel, and replaced by any form-specific navigation links defined for the new form.

To specify the Associated Task Pane, enter the full path and file name of the task pane document (AXL file) that contains the links. For example:

```
\Axiom\Task Panes Library\FormLinks.axl
```

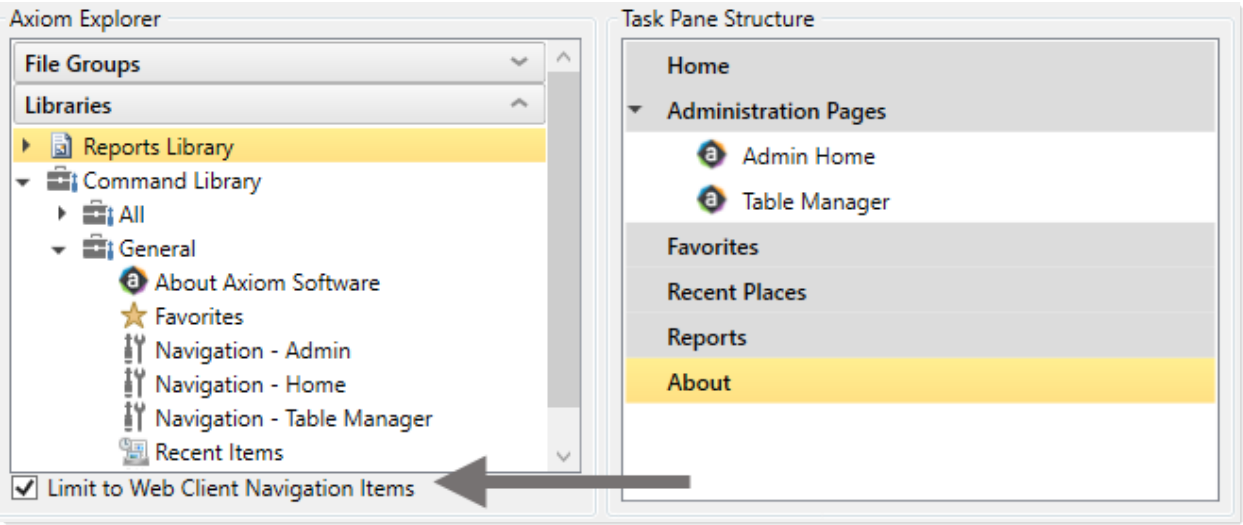
To easily obtain the full path for a file, navigate to that file in the Explorer task pane, then right-click it and select **Copy document path to clipboard**. You can then paste the value into the cell.

The next time you open the form-enabled document after saving, the Associated Task Pane entry will be automatically converted into a system-managed document shortcut (you can tell the difference by the

presence of a `_tid` parameter on the end of the shortcut). If you need to change the entry to point to a different document, simply enter the path as you would have originally, and it will be converted again when you save the file.

Using the task pane editor to define web navigation links

Web navigation links are defined using task pane files. When working in a file that is intended to be used for web navigation links, make sure to enable the setting **Limit to Web Client Navigation Items**, located at the bottom left of the **Edit Task Pane** dialog.



This setting is enabled by default in the WebClientNavigationPane file; you must enable it manually when creating any form-specific navigation files. When enabled, the task pane editor is limited to only showing items that are relevant for use as navigation links. Additionally, the task pane editor is streamlined so that inapplicable settings are not available (such as the Custom Image setting and Display Settings).

The following items can be placed in a task pane used for web navigation links:

- **Text-only items.** These items can be used as expandable/collapsible headers for sets of links. You can nest links using any number of header levels.
- **Form-enabled files.** These files will be opened as forms by default. The available shortcut parameters allow specifying an optional Quick Filter, and whether the file should be opened in the current window or a new window (default is current window).
- **Web reports.** The available shortcut parameters allow specifying whether the report should be opened in the current window or a new window (default is current window).

- **Folders.** The folder structure (including any subfolders) will display in the Navigation panel, automatically showing all web-enabled files that the user has rights to access. Folders without any web-enabled files will not display in the Navigation panel. This provides an easy way to display all relevant contents of a particular folder in the navigation pane.

  When files are shown in the navigation pane via a folder listing, it is not possible to define a Quick Filter for any individual file or specify that the files should open in a new window. These options are only available when the files are added to the task pane individually.

- **Commands.** The commands **About Axiom Software**, **Favorites**, and **Recent Items** can all be used in the Navigation panel, and will automatically display web-applicable content. Additionally, several **Navigation** commands are available to navigate to certain areas of the Web Client.

NOTES:

- For each item in the task pane, you can optionally define display text and a tooltip. If you do not define display text, the default text for the item is used (such as the file name or the folder name).
- You can add shortcuts to top-level items if the item does not have child items, but the task pane editor does not allow dragging and dropping items from the Axiom Explorer pane to the top level. You must use **Add New Item > New top level section** first, then use the **Shortcut Target** property to assign the desired file, folder, or command.
- When the Navigation panel is viewed by a user, if a top-level item has no visible children underneath it (because of items being hidden due to security permissions), then the top-level item will also be hidden.

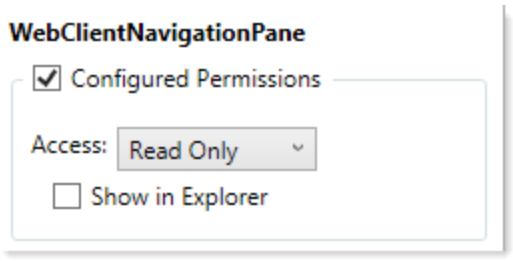
For general information about using the task pane editor, see the *System Administration Guide*.

Security considerations

The links defined in the global file WebClientNavigationPane are available to all users. However, if a user does not have security access to an item listed in the file, that particular link will not display. If all items underneath a text-only header item are hidden due to security reasons, then the header item will be hidden as well.

Users must have security access to any file that defines form-specific navigation links (**Associated Task Pane** files). If a user does not have access to these files, then the links will not display on the navigation panel.

The recommended way to grant this access is to configure the file as **Read Only** and disable **Show in Explorer**. This means that the user will be able to access the file when they open a form that uses it, but otherwise they will not see the file in any file explorer views like the Explorer task pane. The access can be set at the user level or the role level, including on the Everyone role. You may want to store all form-specific navigation task panes in a specific sub-folder and then grant the access at the sub-folder level.


WebClientNavigationPane
Configured Permissions
Access:
Read Only
Show in Explorer

Index

B

C

D

E

F

W

X

Y