Axiom File Setup Guide

Axiom

Version 2020.3

Syntellis
10 S. Wacker Dr., Suite 3375
Chicago, Illinois 60606
847.441.0022
www.syntellis.com

Table of Contents

Introduction

In Axiom, Axiom files are used for all planning and reporting processes. Plan files, templates, drivers, and reports are all built from Axiom files.

Axiom files are controlled Microsoft Excel spreadsheets that support Axiom functionality such as the ability to query data from the database, and save data to the database.

This guide discusses how to set up the various features of Axiom files.

Intended audience

This guide is intended for administrators and other power users who are responsible for creating plan templates, driver files, and reports.

What is covered in this guide?

This guide covers the following aspects of Axiom file setup:

- Creating Axiom queries and data lookups to bring data into Axiom files
- Configuring workbook and worksheet settings such as protection, visibility, and view controls
- Configuring save-to-database processes
- Configuring "refresh" features such as refresh variables and action code processing
- Setting up files for features such as drilling or printing

What is not covered in this guide?

The following related topics are not covered in this guide:

- Calc method libraries and calc method controls. Because calc method libraries can only be used in templates/plan files, they are discussed in the *File Group Administration Guide*.
- Axiom form creation. For information on how to create form-enabled files, see the *Axiom Forms and Dashboards Guide*.
- Axiom function syntax. For reference information on all Axiom functions, see the *Axiom Function Reference*.

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

Axiom 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 functionality is virtually identical in both environments.

When designing Axiom files for use in the Windows Client, keep in mind that some Microsoft Excel functionality is not supported in the Windows Client, or may not work in exactly the same way. For more information, see Windows Client Design Considerations.

2

Axiom File Overview

Axiom files are enhanced spreadsheets used for reports, templates, plan files, utilities, and driver files. Each Axiom file has a Control Sheet and supports special features, such as:

- Bringing data from the database into Axiom files
- Saving data from Axiom files to the database
- Inserting or changing rows by using calc method libraries (available in file groups only)
- Enhanced navigation and view controls throughout the workbook
- Inherited settings for freezing panes, sheet protection, and other spreadsheet options

The defining characteristic of an Axiom file is a Control Sheet where many of the settings for that file are defined. Other settings are defined in the individual sheets through use of control columns, control rows, and reserved feature tags. This guide discusses how to set up Axiom file features.

With a few exceptions, Axiom file features and Control Sheet settings apply to all Axiom file types. For example, plan files and report files have the same base functionality. The same features are used to bring data into the file, and to save data to the database. The difference is the context and purpose of the file, not its inherent functionality.

Control Sheets

Every Axiom file has a Control Sheet (tab name: **Control_Sheet**). The Control Sheet contains settings for each sheet in the workbook.

Control Sheet settings define the following:

- Save to database options
- Sheet display and protection options
- Sheet filters
- Axiom queries

Control Sheet settings can be configured manually on the sheet, or you can use the Sheet Assistant task pane to edit most common settings. Some settings, such as save-to-database settings, are only available on the Control Sheet.

Most sheets in an Axiom file are set up on the Control Sheet. However, if a sheet does not need to use any of these Axiom file features, then that sheet does not need to be configured on the Control Sheet.

Control sheet settings will be referenced throughout this guide. For a detailed reference of the settings that are available on the Control Sheet, see Control Sheet Settings.

Viewing the Control Sheet

By default, the Control Sheet is automatically visible or hidden based on the current user's **Allow Sheet Assistant** security permission for the file (as configured on the **Files** tab of security):

- If you are an administrator or you have the **Allow Sheet Assistant** permission, *and* you open the document as read/write, then the Control Sheet is visible. This behavior is intended to provide report writers with easy access to configuration settings on the Control Sheet.
- Otherwise, the Control Sheet is hidden.

If instead you want the Control Sheet to start as hidden for all users, you can use the **Hide Control Sheet on open** setting. This setting is located on the Control Sheet, in the **Workbook Options** section.

- If **Hide Control Sheet on open** is set to **On**, then the Control Sheet is hidden for all users when the file is opened.
- Otherwise, the default behavior applies, where the Control Sheet is automatically visible or not based on Sheet Assistant security permissions. This default behavior also applies if the **Hide Control Sheet on open** setting is blank or not present.

If **Hide Control Sheet on open** is not enabled, but the Control Sheet is still being hidden for all users when the file is opened, this may be happening because the file was saved in an earlier version where the default behavior was different (or because the Control Sheet was manually hidden using Excel functionality). In this case, you can open the file, unhide the Control Sheet, and then save the file. The next time the file is opened, the Control Sheet should behave as previously described.

NOTE: In plan files, the Control Sheet is always hidden by default and the workbook is always protected, regardless of the Control Sheet settings. It is rarely necessary to access the Control Sheet in plan files, because any configuration changes for plan files need to be made in the template.

If the Control Sheet is hidden, and the workbook is protected, users with the appropriate permissions can unhide it as follows:

- Users with access to the Sheet Assistant can double-click a property name in the Sheet Assistant to be taken to the corresponding property in the Control Sheet. For example, you can double-click **Query Name** to be taken to the name of the Axiom query on the Control Sheet. This will also unhide the Control Sheet if it is hidden.
- Users with permission to unprotect workbooks can manually unprotect the file and unhide the sheet. Administrators can also use the **Show Everything** feature on the Axiom Designer ribbon tab.

Bringing data into Axiom files

Axiom supports three different ways to query data from the Axiom database and bring it into Axiom files, such as plan files and reports:

- Axiom queries
- Lookup data sources
- Axiom functions

All three features can be used in any Axiom file, however, each feature has its own strengths and weaknesses, as well as a performance impact.

When choosing which feature to use, you should give preference in the order listed above. Axiom queries are most often the very best method for returning data and should be used whenever possible. If an Axiom query cannot be used for a particular use case, then a Lookup data source is the next best option. Axiom functions should only be used if the data cannot be returned using either of the previous two methods. For more information, see the summaries of each feature below.

Once a file has been set up to query data, various additional features can be used to further impact the data that is brought into the file, such as data filters, refresh forms, and data conversion.

The following summary explains the basics of each query method, to help understand when to use them.

Axiom queries

Description	Axiom queries are the primary method of bringing data into Axiom files. Axiom queries have many advantages over other query methods, including: • Flexible setup that can return various combinations of data in many different formats • Configurable refresh behavior that can either update existing static rows of data, or dynamically build out sections of data including applying formatting and formulas via Excel-based calc methods • Efficient data query method that only runs when it is explicitly triggered, resulting in improved file performance • Many basic and advanced options to fine tune the data query and the results Axiom queries are set up using a combination of Control Sheet settings and tags in the sheet. When a refresh occurs, the resulting data from the query is brought into a designated data range within the sheet. Axiom queries can return data from any client-defined table—including returning data from multiple tables within the same query. Axiom queries can also return data from any system table (one table per query).
When to Use It	Axiom queries should always be the first option you consider when you need to bring data into a file. Axiom queries are highly efficient and very flexible to meet a variety of reporting and data query needs.
More Information	See Axiom Queries.

Data Lookups

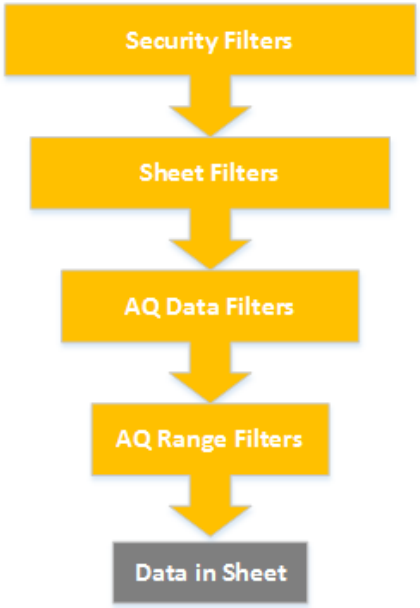
Description	Data lookups are data queries that can be configured to execute at specific times. These queries can be used to bring in relatively "static" points of data that do not need to be continually refreshed while the user is working in the file (as they would be if using Axiom functions). For example, in a plan file, data lookups can be used to bring in static data such as the department description and the file group name. This type of data usually does not change, and even if it does change, it is not essential to reflect that change in the current file session. Data lookups support a variety of execution options to meet various needs. They can be configured to run automatically whenever a file is refreshed, or they can be configured to run only at specific times—such as on file open, or after a particular Axiom query is executed. You can also optionally expose the ability for end users to execute data lookups as needed, using the **Execute Data Lookups** command. Data lookups are defined using a DataLookup data source. Each row in the data source defines a data point to be queried and returned into the designated column of the data source. These values can then be referenced throughout the file as needed.
When to Use It	Data lookups are primarily intended to be a substitute for Axiom functions, to return specific points of data into a file. Use of data lookups over Axiom functions can greatly improve file performance, because the database query used for data lookups is more efficient, and because the execution of data lookups is completely controllable. Currently, data lookups can be used to return the same data as the following Axiom functions: • GetData • GetFeatureInfo • GetFileGroupProperty • GetFileGroupVariable • GetFileGroupVariableEnablement • GetFileGroupVariableProperty • GetPlanItemValue • GetSecurityInfo • GetTableInfo • GetUserInfo
More Information	See Data Lookups.

Axiom Functions

Description	Axiom functions bring data into specific cells (the cell containing the function). They can be used directly within Axiom files to create fixed sections of data, or they can be used within calc methods and therefore populated into Axiom query data ranges. Axiom functions can be used in combination with Excel functions or with other Axiom functions (nested functions). The primary function used to return data is GetData. GetData can return data from any client-defined table as well as most system tables. Each GetData function returns a single value into a single cell. This value can be a single data point in the table, or it can be the result of summing a set of data (or using other alternate aggregations such as Average). Other functions are available to return certain specialized data, such as process and security information, and file group properties.
When to Use It	Due to their slower performance, Axiom functions should be considered as the "last resort" for a particular data query need. Function-based data queries are much less efficient than the other two query methods. The same data that can be returned in a single Axiom query execution may instead require thousands of individual data queries when using functions. This difference in efficiency quickly adds up as functions are used in a file, and can significantly impact the file performance. Function-based data queries are also less controllable. Function updates may be triggered continuously as users work in the spreadsheet, causing dependent and volatile functions to fire and therefore impact file performance. In rare cases this kind of continuous update may be necessary, but in most cases it is not and a more controllable query method should be used instead.
More Information	See the separate document *Axiom Function Reference*.

Filtering data in an Axiom file

When an Axiom file is refreshed and data is brought into the file, various levels of filters can be applied to the data. This section provides an overview to the various filters that may apply when data is queried in a file.



Order of filter application in an Axiom file

All of the following filters are combined to result in the final set of data brought into a sheet:

1. **User security filters:** Each user can have read filters defined for each table or table type, to control what data the user can access. These filters are defined in Axiom Security, at the user and/or role level.

   User security filters are always applied, whenever queries are made to the applicable tables. Because of this, different users may see different data (or no data) when they refresh the same file.

   For more information on table security filters, see the *Security Guide*.

2. **Sheet filters:** Each sheet in an Axiom file can have one or more sheet filters, to filter the data being brought into the sheet. Sheet filters apply to all Axiom queries and GetData functions on the sheet.

   Sheet filters can be defined and saved on the Control Sheet of a file, or they can be temporarily applied by use of the Quick Filter feature or a GetDocumentHyperlink URL.

   GetData functions and data lookups can use an optional parameter to ignore any sheet filters. Sheet filters always apply to Axiom queries.

   For more information on sheet filters, see Defining sheet filters.

3. **Axiom query filters:** The following filter options only apply to Axiom queries.

    - **Data filter:** Each Axiom query defined on a sheet can have a data filter that applies to the entire query. The data filter is defined on the Control Sheet.
    - **Data range filters:** Each data range of an Axiom query can have a filter that determines what data from the query is brought into that particular data range.

      Data ranges are defined by placement of `[AQ#]` and `[Stop]` tags. The filter is appended to the AQ tag, such as: `[AQ1;DEPT.Region='North'].`

      If a data record is returned by the Axiom query, but there is no place for it on the sheet due to use of data range filters, then the data record is not brought into the sheet.

    Axiom queries support a few other specialized features that can impact the data to be returned, such as using column filters to limit the data coming in a particular column, or limiting query results to the "top n" (such as top 10). For more information on Axiom query filters and other special features, see Axiom Queries.

3

Display and Protection Options

You can configure various worksheet and workbook options for display and protection, such as:

- Freeze panes
- View controls
- GoTo bookmarks
- Worksheet visibility
- Worksheet and workbook protection

Setting up views for Axiom files

Spreadsheet Axiom files can use multiple types of *views* that specify which rows and columns in a sheet are hidden or shown. Views can also be used to set row and column sizing.

You might have a single view in a sheet for formatting purposes only, or you might have several different views so that users can switch between different views of the data. For example, you might want users to be able to switch between viewing the data by quarters or by months. Or you might use views to hide optional "detail" rows or columns that users can then expose as needed. The ability to change which rows or columns are shown at any one time can help make complex and detailed spreadsheets easier to read.

You can set default views to apply automatically when the file is opened. Once a file is opened, users can select and apply views by using the **Change View** menu on the default Axiom ribbon tab (or by using the **Change View** command in any custom task pane or ribbon tab).

Axiom provides three types of views:

- **Sheet Views** control the overall view of a sheet. They can be used to set row and column sizing, and to flag certain rows and columns as hidden. Only one sheet view can be active at a time.
- **Column Views** can be used to toggle sets of columns as shown or hidden. Multiple column views can be active at a time.
- **Row Views** can be used to toggle sets of rows as shown or hidden. Multiple row views can be active at a time.

In general, sheet views are intended for cases where you want to affect the entire sheet and provide a small handful of different viewing options. For example, you might have a Summary sheet view that shows only the high-level data and then a Detail sheet view that exposes the detailed data. Depending on what the user is doing at any one time, they might only need to see the high-level data or they might need to dive into the details. The user can switch between the two views as needed.

Column and row views are intended for cases where you want the user to be able to dynamically toggle certain rows and columns as shown or hidden. This is a similar experience to Excel's AutoFilter feature, where users can view a list of items and select which ones they want to show, in any combination. For example, you could have column views set up to show columns by quarter. The user could choose to show only the columns for Q1, or show both Q1 and Q2, or show all four quarters, or show any combination of quarters.

You can use any combination of sheet views, column views, and row views in a sheet. Column views and row views do not have any area of overlap, but sheet views may overlap with column or row views. If you are using sheet views with column or row views, be sure that you understand how the views will interact, and be sure to test the setup. For more information, see Using multiple view types in a sheet.

Defining sheet views for a sheet

You can define one or more sheet views for a sheet, in order to:

- Set sizing for columns and rows in a sheet
- Hide specific columns and rows in a sheet
- Set the active cell for a sheet

A sheet can have a single view just for formatting purposes, or it can have multiple views that emphasize different information. For example, a sheet might have one view that displays data by quarters, and another view that displays data by months. Or one view that shows only high-level summary data, and another view that exposes the detailed data.

Sheet views are defined by creating the following tags in the sheet:

- A View tag to define the name of the sheet view, and to designate the location of the view control row and control column. You can also optionally specify default row and column sizing for the sheet view.
- Hide tags to flag certain columns and rows to be hidden when the sheet view is applied. These tags are placed in the view control row and control column.
- Set tags to set sizing for specific columns and rows when the sheet view is applied. These tags are placed in the view control row and control column.
- Page break tags to set page breaks when the sheet view is used in conjunction with a Print tag. These tags are placed in the view control row and control column.

Sheet view tag summary

Tag Type	Tag Syntax
Primary tag	`[View; ViewName; DefaultColWidth; DefaultRowHeight; ActiveCell]`
Hide row / column tags	`[Hiderow;Pagebreak]` `[Hidecolumn;Pagebreak]`
Set row / column tags	`[Setrow;Height;Pagebreak]` `[Setcolumn;Width;Pagebreak]`
Pagebreak row / column tag	`[Pagebreak]`

Defining the primary View tag in a sheet

To define a sheet view, place the following tag in any cell within the first 500 rows of the sheet:

`[View; ViewName; DefaultColWidth; DefaultRowHeight; ActiveCell]`

The View tag uses the following parameters:

Item	Description
ViewName	The name of the sheet view. The sheet view name is the name that users will select to apply this particular view. The name can also be used to apply a default sheet view when the file is opened. This name must be unique across all views defined in the sheet, including column views and sheet views.
DefaultColWidth	Optional. Specifies the default column width for the sheet view. The width specified here will be used for all columns except hidden columns, or columns where the width is set using individual SetColumn tags. You can type a number for the width, or use the keyword `Autofit`.
DefaultRowHeight	Optional. Specifies the default row height for the sheet view. The height specified here will be used for all rows except hidden rows, or rows where the height is set using individual SetRow tags. You can type a number for the height, or use the keyword `Autofit`.

Item	Description
ActiveCell	Optional. Specifies a cell location for the cursor when the sheet view is applied. The active cell is only used when the view is applied as an **Initial Dynamic View** or when a user explicitly applies the view using **Change View**. It is not reapplied after system processes such as running Axiom queries or inserting calc methods. **NOTE:** If the view is configured to be applied as the default view when the file is opened, the active cell defined for the view takes priority over the active cell defined for the sheet (on the Control Sheet).

The row where the View tag is placed becomes the view control row, and the column where the view tag is placed becomes the view control column. Multiple sheet views can share a control row or a control column if appropriate.

Examples

```
[View;Budget Summary]
```

This defines a sheet view named Budget Summary, with no default column or row sizing.

```
[View;Budget Detail;Autofit;16]
```

This defines a sheet view named Budget Detail, with the default column width set to autofit, and the default row height set to 16.

```
[View;Budget Summary;;;F25]
```

This example is the same as the first example, except that the cursor will be placed at cell F25 when the sheet view is applied. Note that omitted parameters must be delimited with "empty" semicolons.

NOTES:

- View tags are not ignored for purposes of the autofit settings. This means that the column containing the View tag may be quite wide due to the length of the View tag itself. Once you have finished setting up the view, you may want to set the cell containing the View tag to a small font size, such as 1pt, so that the tag does not affect autofit.
- The primary View tag must be located in the first 500 rows of the sheet.
- View tags can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Hiding columns or rows

If you want a column or row to be hidden when a sheet view is applied, you must mark the column or row with a Hide tag.

To mark a row as hidden, place the following tag in the control column for the sheet view:

`[Hiderow]`

To mark a column as hidden, place the following tag in the control row for the sheet view:

`[Hidecolumn]`

If a column or row is marked to be hidden, this overrides any default width or height set in the View tag, and the column or row is hidden using standard Excel functionality.

NOTES:

- Hide tags can be placed within formulas if desired. For example, you can use formulas to dynamically determine whether a certain row or column should be hidden.
- `[Hidecol]` is also recognized as a valid tag.
- You can optionally append the keyword `Pagebreak` to the Hide tag in order to set a page break when printing. For example: `[Hiderow;Pagebreak]`. See Setting page breaks for use with Print views.

Setting individual column width or row height

If you want to set the height of an individual column or the width of an individual row when a sheet view is applied, you must mark the column or row with a Set tag. If an individual column or row is marked with a Set tag, this overrides any default width or height set in the View tag.

To set the height of a row, place the following tag in the control column for the sheet view:

`[Setrow;Height]`

To set the width of a column, place the following tag in the control row for the sheet view:

`[Setcolumn;Width]`

To define the height or the width in the tag, specify a number or use the keyword `Autofit`.

NOTES:

- The Set tag can be placed within a formula if desired. For example, you can use formulas to dynamically determine the height or width, or whether the size should be set at all.
- Set tags are not ignored for purposes of autofit settings. You may want to set the cells containing Set tags to a small font, such as 1pt, so that the tags do not affect autofit.
- You can optionally append the keyword `Pagebreak` to the Set tag in order to set a page break when printing. For example: `[Setcolumn;15;Pagebreak]`. See Setting page breaks for use with Print views.
- `[Setcol]` is also recognized as a valid tag.

Setting page breaks for use with Print views

If the sheet view will be used with a Print view, then you can specify page breaks on certain rows and columns. These tags are only honored when the sheet is printed using a Print tag that references the sheet View tag. The page break tags are ignored when simply applying the view to the sheet. For more information on using Print tags, see Setting up print views for Axiom files.

To set a page break on a particular row or column, place the Pagebreak tag in the View control column or row by itself, or as an optional parameter to a Hide or Set tag. For example:

```
[Pagebreak]

[Hiderow;Pagebreak]

[Setcolumn;15;Pagebreak]
```

Page breaks are inserted *before* the column or row that is flagged with a Pagebreak tag. This means that the next page will start with the flagged row or column (unless the Pagebreak tag is within a Hide tag, in which case the next page starts with the next visible column or row).

Sheet view behavior

When a file is opened, no sheet views are active by default. This means that no sheet view settings will be applied, and all columns and rows will be visible (unless they are hidden by other means). If you want to automatically activate a sheet view when the file is opened, you can use the **Initial Dynamic View** setting on the Control Sheet. For more information, see Setting the default views for a sheet.

When a sheet view is made active—whether by default on file open, or by using the **Change View** command—the following occurs:

- All rows and columns that are marked with Hide tags in the selected view are hidden.
- All other rows and columns are shown, including rows and columns that have been manually hidden using Excel functionality.

  Exception: If a column or row is currently hidden due to an active column view or row view, that column or row remains hidden. When sheet views overlap with column and row views, the Hide tags in the sheet view are always honored, but otherwise the column and row view settings remain active.
- Row heights and column widths are set based on the default settings in the selected View tag and in any individual Set tags. If no width or height is defined in the sheet view for a column or row, then its width or height is left as is.
- The cursor is placed in the active cell for the view, if one has been defined for the view.
- The Freeze Panes settings defined on the Control Sheet (if applicable) are reapplied.
- Spreadsheet groupings in the sheet are expanded.

When a sheet view is made active, the name of the sheet view is written to the **Current Dynamic View** cell on the Control Sheet. This field is system-maintained and is used to keep track of the current sheet view so that it can be automatically reapplied after certain file processes (such as after running an Axiom query, or inserting a calc method). If the sheet is not already configured on the Control Sheet, activating a sheet view will cause it to be.

Only one sheet view can be active at any one time. If a sheet view is currently active and then a user selects a different sheet view from the **Change View** menu, the settings for the existing view are now ignored and the settings for the new view are applied.

Defining column views in a sheet

You can define column views in a sheet, to allow users to toggle certain columns as shown or hidden.

For example, you may have twelve months of data in a sheet and you want users to be able to show or hide these columns by quarters. You can set up these columns with column view tags, and then users can specify which quarters are currently visible in the sheet by toggling these column view tags off or on.

Column views are additive by default. Unlike sheet views, where the user must switch between one view or another, any number of column views can be active at the same time. Continuing the previous example, if a user wants to see the columns for both Q1 and Q3, they can choose to activate both of those column views at the same time. However, if desired it is also possible to configure a ColumnView tag so that only one column view within that control row can be active at a time.

Column views are defined by creating the following tags in the sheet:

- A ColumnView tag to designate the control row for this set of column views. The tag can also be used to define an overall group name for the column views, and to optionally enforce single selection of column views.
- Column tags to mark columns to be shown or hidden, and to define the column view names. These tags are placed in the control row.

Column view tag summary

Tag Type	Tag Syntax
Primary tag	`[ColumnView; GroupName; SelectionType]`
Column tags	`[Column; ViewName]`

Defining the ColumnView tag in a sheet

To define a set of column views, place the following tag in any cell within the first 500 rows of the sheet. The row where the ColumnView tag is placed becomes the column view control row.

`[ColumnView; GroupName; SelectionType]`

The ColumnView tag uses the following parameters:

Item	Description
GroupName	Optional. A top-level group name for all column views defined in this control row. • If omitted, then all view names defined in the Column tags will display directly on the **Change View** menu. • If defined, then all view names defined in the Column tags will belong to the group name. The specific behavior of the menu depends on whether the SelectionType is set to allow multiple selection or single selection. For more information, see Configuring view display on the Change View menu. This name must be unique across all views defined in the sheet, including row views and sheet views.
SelectionType	Optional. Specifies whether the column views in this control column are additive (multiple selection), or whether only one column view can be active at a time (single selection). • If omitted, then multiple column views can be active at a time. This is the default behavior. • Otherwise, use the keyword `Single` to specify that only one column view (from this control row) can be active at a time. **NOTE:** If you enable single-selection, then you must also define a group name. The column view names affected by the single-selection behavior are displayed underneath the group name, so that the user can understand that selecting one of these items causes the other items in the group to become inactive. The group name should attempt to make this behavior clear—for example, it should say something like "Select a Quarter to Show" versus "Select Quarters to Show."

Examples

```
[ColumnView]
```

This tag simply designates the control row for the column views. The specific names defined in the individual Column tags display directly on the menu.

```
[ColumnView;Show Columns]
```

In this example, a group name has been defined for the tag. This name will display as the first level on the **Choose View** menu instead of the individual column view names. The user can then select one or more column views within this group.

```
[ColumnView;Show a Column;Single]
```

In this example, the text "Single" is used in the third parameter to specify that only one column view within this group can be active at a time.

NOTES:

- The primary ColumnView tag must be located in the first 500 rows of the sheet.
- ColumnView tags can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Marking columns to include in column views

Within the view control row, use the following tag to mark each column that you want users to be able to toggle shown or hidden:

```
[Column; ViewName]
```

ViewName defines the name that users will select to toggle this column shown or hidden. The name can also be used to apply a default column view when the file is opened.

Multiple columns in the same ColumnView control row can use the same view name so that those columns are toggled as a unit. However, the name cannot be used in any other view control.

When a column view name is toggled active, columns marked with that name are shown. When a column view name is toggled inactive, columns marked with that name are hidden. Any columns in the control row that are *not* marked with Column tags are unaffected by the column view.

Example

	A	B	C	D	E	F	G	H	I
3									
4		[ColumnView]		[Column;Q1]	[Column;Q1]	[Column;Q1]	[Column;Q2]	[Column;Q2]	[Column;Q2]
5									

Columns D-F belong to the column view named Q1, and columns G-I belong to the column view named Q2. When Q1 is active, columns D-F are shown; when Q1 is inactive, then columns D-F are hidden. If both Q1 and Q2 are active, then columns D-I are shown. Column C is not marked with a Column tag and therefore is unaffected by the column view.

You can define multiple view names within a single Column tag, separated by commas. For example:

```
[Column; All, Q1]
```

This column belongs to both the Q1 column view and the All column view. This approach could be used in the Column tags for the previous example so that users have a way to toggle all of the quarters visible at once.

Column view behavior

When a file is opened, column views are automatically applied but inactive by default. This means that all columns that belong to a column view will be hidden (because the column views are not active). If you want to automatically activate one or more column views when the file is opened (so that the columns will be visible), you can use the **Initial Dynamic Column Views** setting on the Control Sheet. For more information, see Setting the default views for a sheet.

When a column view is made active—whether by default on file open, or by using the **Change View** command—the following occurs:

- All columns marked with the designated column view name are shown.

  Exception: If a sheet view is also active in the sheet, and one of the columns belonging to the column view is flagged to be hidden by the sheet view, then the column will be hidden. If a sheet view and column view overlap, the Hide tag in the sheet view takes priority.

- By default, all other marked columns in the same ColumnView control row are left as is (shown or hidden). However, if the ColumnView tag is configured as single-selection, then all other marked columns in the control row are hidden (and their associated views made inactive).

- The Freeze Panes settings defined on the Control Sheet (if applicable) are reapplied.

When a column view is made active, the name of the column view is written to the **Current Dynamic Column Views** cell on the Control Sheet. Multiple active column views are separated by commas. This field is system-maintained and is used to keep track of the currently active column views so that they can be automatically reapplied after certain file processes (such as after running an Axiom query, or inserting a calc method). If the sheet is not already configured on the Control Sheet, activating a column view will cause it to be.

Multiple column views can be active at a time. All columns belonging to all active column views will be shown (unless a column is hidden by an active sheet view, as described previously).

Using multiple ColumnView tags

You can define multiple ColumnView tags in a sheet, to define different sets of column views. You might do this for the following reasons:

- You want one set of column views to allow multiple-selection, while the other set only allows single selection. Because this is configured on the ColumnView tag, you must use multiple ColumnView tags.

- You want to use different group names for different sets of column views. Because this is configured on the ColumnView tag, you must use multiple ColumnView tags.

- You want a column to belong to more than one column view. However, keep in mind that you may be able to accomplish this more easily by defining multiple column view names within a single Column tag.

Although each individual ColumnView tag may be configured as multiple-selection or single-selection, if you have multiple ColumnView tags then these different sets of column views are always additive. Enabling single-selection for a particular ColumnView tag only controls the column view selections for that particular control row. The user can still select column views from other ColumnView control rows if they exist, and those columns will be shown even if they would have otherwise been hidden by the single-selection ColumnView.

NOTE: If you have multiple `[ColumnView]` tags in a sheet, make sure that the column view names defined in the `[Column]` tags are unique within each control row. All columns flagged with the same view name will be treated as belonging to the same view, even if the columns are in different `[ColumnView]` control rows.

Defining row views in a sheet

You can define row views in a sheet, to allow users to toggle certain rows as hidden or shown.

For example, you may have multiple row "blocks" within a sheet (such as for different regions or departments), and you want users to be able to show or hide these row blocks individually. You can set up these rows with row view tags, and then users can specify which row blocks are currently visible in the sheet by toggling these row view tags off or on.

Row views are additive by default. Unlike sheet views, where the user must switch between one view or another, any number of row views can be active at the same time. Continuing the previous example, if a user wants to see the row blocks for both Region West and Region East, they can choose to enable both of those row views at the same time. However, if desired it is also possible to configure a RowView tag so that only one row view within that control column can be active at a time.

Row views are defined by creating the following tags in the sheet:

- A RowView tag to designate the control column for this set of row views. The tag can also be used to define an overall group name for the row views, and to optionally enforce single selection of row views.
- Row tags to mark rows to be shown or hidden, and to define the row view names. These tags are placed in the control column.

Row view tag summary

Tag Type	Tag Syntax
Primary tag	`[RowView; GroupName; SelectionType]`
Row tags	`[Row; ViewName]`

Defining the RowView tag in a sheet

To define a set of row views, place the following tag in any cell within the first 500 rows of the sheet. The column where the RowView tag is placed becomes the view control column.

`[RowView; GroupName; SelectionType]`

The RowView tag uses the following parameters:

Item	Description
GroupName	Optional. A top-level group name for all row views defined in this control column. • If omitted, then all view names defined in the Row tags will display directly on the **Change View** menu. • If defined, then all view names defined in the Row tags will belong to the group name. The specific behavior of the menu depends on whether the SelectionType is set to Single. For more information, see Configuring view display on the Change View menu. This name must be unique across all views defined in the sheet, including row views and sheet views.
SelectionType	Optional. Specifies whether row views are additive, or whether only one row view can be active at a time. • If omitted, then multiple row views can be active at a time. This is the default behavior. • Otherwise, use the keyword `Single` to specify that only one row view (from this control column) can be active at a time. **NOTE:** If you enable single-selection, then you must also define a group name. The column view names affected by the single-selection behavior are displayed underneath the group name, so that the user can understand that selecting one of these items causes the other items in the group to become inactive. The group name should attempt to make this behavior clear—for example, it should say something like "Select a Region to Show" versus "Select Regions to Show."

Examples

```
[RowView]
```

This tag simply designates the control column for the row views. The specific names defined in the individual Row tags display directly on the menu.

```
[RowView;Show Rows]
```

In this example, a group name has been defined for the tag. This name will display as the first level on the **Choose View** menu instead of the individual row view names.

```
[RowView;;Single]
```

In this example, the text "Single" is used in the third parameter to specify that only one row view (from this control column) can be active at a time. Note that the omitted second parameter must be delimited with an "empty" semicolon.

NOTES:

- The primary RowView tag must be located in the first 500 rows of the sheet.
- RowView tags can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Marking rows to include in row views

Within the view control column, use the following tag to mark each row that you want users to be able to toggle shown or hidden:

```
[Row; ViewName]
```

ViewName defines the name that users will select from the **Change View** menu to toggle this row shown or hidden. The name can also be used to apply a default row view when the file is opened.

Multiple rows in the RowView control column can use the same view name so that those rows are toggled as a unit. However, the name cannot be used in any other view control.

When a row view name is toggled active, rows marked with that name are shown. When a row view name is toggled inactive, rows marked with that name are hidden. Any rows in the control column that are *not* marked with Row tags are unaffected by the row view.

Example

	A	B
2		
3		[RowView]
4		
5		[Row;Section A]
6		[Row;Section A]
7		[Row;Section A]
8		[Row;Section B]
9		[Row;Section B]
10		[Row;Section B]

Rows 5-7 belong to the row view named Section A, and rows 8-10 belong to the row view named Section B. When Section A is active, rows 5-7 are shown; when Section A is inactive, then rows 5-7 are hidden. If both Section A and Section B are active, then rows 5-10 are shown. Row 4 is not marked with a Row tag and therefore is unaffected by the row view.

You can define multiple view names within a single Row tag, separated by commas. For example:

```
[Row; Section A,All]
```

This row belongs to both the Section A row view and the All row view. This approach could be used in the Row tags for the previous example so that users have a way to toggle all of the sections visible at once.

Row view behavior

When a file is opened, row views are automatically applied but inactive by default. This means that rows that belong to row views will be hidden (because the row views are not active). If you want to automatically activate one or more row views when the file is opened (so that the rows will be visible), you can use the **Initial Dynamic Row Views** setting on the Control Sheet. For more information, see Setting the default views for a sheet.

When a row view is made active—whether by default on file open, or by using the **Change View** command—the following occurs:

- All rows marked with the designated row view name are shown.

  Exception: If a sheet view is also active in the sheet, and one of the rows belonging to the row view is flagged to be hidden by the sheet view, then the row will be hidden. If a sheet view and row view overlap, the Hide tag in the sheet view takes priority.

- By default, all other marked rows in the same control column are left as is (shown or hidden). However, if the RowView tag is configured as single-selection, then all other marked rows in the control column are hidden (and their associated row views made inactive).
- The Freeze Panes settings defined on the Control Sheet (if applicable) are reapplied.

When a row view is made active, the name of the row view is written to the **Current Dynamic Row Views** cell on the Control Sheet. Multiple active row views are separated by commas. This field is system-maintained and is used to keep track of the current row views so that they can be automatically reapplied after certain file processes (such as after running an Axiom query, or inserting a calc method). If the sheet is not already configured on the Control Sheet, activating a row view will cause it to be.

Multiple row views can be active at a time. All rows belonging to all active row views will be shown (unless a row is hidden by an active sheet view, as described previously).

Using multiple RowView tags

You can define multiple RowView tags in a sheet, to define different sets of row views. You might do this for the following reasons:

- You want one set of row views to allow multiple-selection, while the other set only allows single selection. Because this is configured on the RowView tag, you must use multiple RowView tags.
- You want to use different group names for different sets of row views. Because this is configured on the RowView tag, you must use multiple RowView tags.

- You want a row to belong to more than one row view. However, keep in mind that you may be able to accomplish this more easily by defining multiple row view names within a single Row tag.

Although each individual RowView tag may be configured as multiple-selection or single-selection, if you have multiple RowView tags then these different sets of row views are always additive. Enabling single-selection for a particular RowView tag only controls the row view selections for that particular control column. The user can still select row views from other RowView control columns if they exist, and those rows will be shown even if they would have otherwise been hidden by the single-selection RowView.

NOTE: If you have multiple `[RowView]` tags in a sheet, make sure that the row view names defined in the `[Row]` tags are unique within each control column. All rows flagged with the same view name will be treated as belonging to the same view, even if the rows are in different `[RowView]` control columns.

Using multiple view types in a sheet

You can use multiple view types in the same sheet to accomplish different goals. When using multiple view types, it is important to understand the interaction between sheet views and row / column views.

A sheet view *hides* marked rows and columns when it is activated, whereas row / column views *show* rows and columns when they are activated. If there are points of overlap—where a row or column is marked as hidden by a sheet view, but shown by a row / column view—the sheet view takes precedence.

Consider the following example with both sheet views and column views.

- This example has two sheet views, All Detail and Summary. All Detail does not hide any columns, whereas Summary hides the columns for the individual months.
- This example has a single ColumnView tag with a group name of Select Quarters. This control row defines four column views representing four quarters: Q1, Q2, Q3, and Q4. (The views for Q3 and Q4 are not shown in the screenshot for space considerations.)

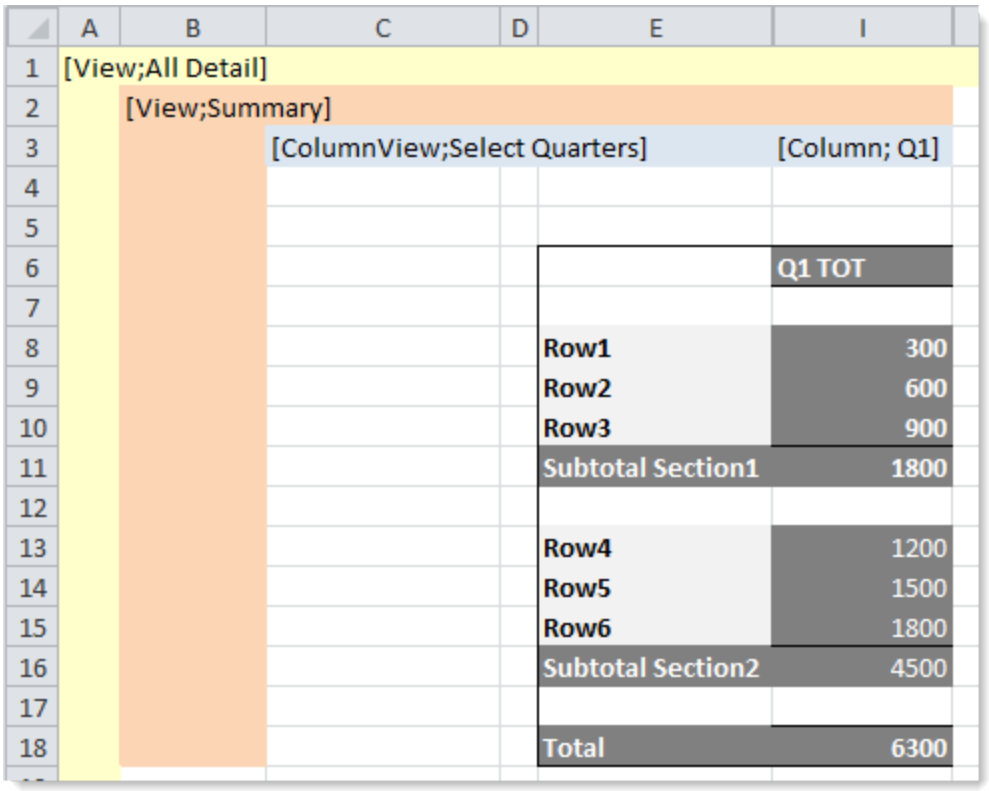
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	[View;All Detail]												
2		[View;Summary]				[HideCol]	[HideCol]	[HideCol]		[HideCol]	[HideCol]	[HideCol]	
3			[ColumnView;Select Quarters]			[Column; Q1]	[Column; Q1]	[Column; Q1]	[Column; Q1]	[Column; Q2]	[Column; Q2]	[Column; Q2]	[Column; Q2]
4													
5													
6						M1	M3	M3	Q1 TOT	M4	M5	M6	Q2 TOT
7													
8					Row1	100	100	100	300	100	100	100	300
9					Row2	200	200	200	600	200	200	200	600
10					Row3	300	300	300	900	300	300	300	900
11					Subtotal Section1	600	600	600	1800	600	600	600	1800
12													
13					Row4	400	400	400	1200	400	400	400	1200
14					Row5	500	500	500	1500	500	500	500	1500
15					Row6	600	600	600	1800	600	600	600	1800
16					Subtotal Section2	1500	1500	1500	4500	1500	1500	1500	4500
17													
18					Total	2100	2100	2100	6300	2100	2100	2100	6300

If All Detail is the active sheet view, then the column views can be toggled as shown or hidden without restriction, because there is no overlap between the views. When column view Q1 is active, those columns will be shown. When column view Q1 is inactive, those columns will be hidden.

	A	B	C	D	E	F	G	H	I
1	[View;All Detail]								
2		[View;Summary]				[HideCol]	[HideCol]	[HideCol]	
3			[ColumnView;Select Quarters]			[Column; Q1]	[Column; Q1]	[Column; Q1]	[Column; Q1]
4									
5									
6						**M1**	**M3**	**M3**	**Q1 TOT**
7									
8					**Row1**	100	100	100	300
9					**Row2**	200	200	200	600
10					**Row3**	300	300	300	900
11					**Subtotal Section1**	600	600	600	1800
12									
13					**Row4**	400	400	400	1200
14					**Row5**	500	500	500	1500
15					**Row6**	600	600	600	1800
16					**Subtotal Section2**	1500	1500	1500	4500
17									
18					**Total**	**2100**	**2100**	**2100**	6300

All Detail as active sheet view, and Q1 as only active column view

If Summary is the active sheet view, then the columns hidden by that sheet view are always hidden, regardless of whether the column view is active. When column view Q1 is made active, now only the "Q1 TOT" column will be shown—the monthly columns will be hidden because they are hidden by the sheet view. However if column view Q1 is active when you switch from sheet view Summary to All Detail, then all columns in that view will now be shown again because the All Detail sheet view is not hiding them.



Summary as active sheet view, and Q1 as only active column view

It is important to understand that the sheet views do not completely override the column views. For example, imagine that column view Q2 is inactive, meaning that all of those columns are currently hidden. If you switch from sheet view Summary to All Detail, the columns in Q2 will not suddenly be shown. The column views retain their active or inactive status when the sheet views change. The only thing that changes is whether or not the new sheet view is forcing a column to be hidden.

Configuring view display on the Change View menu

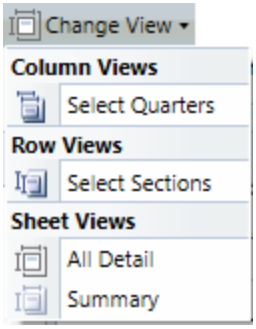
There are several options to configure the display of views on the **Change View** menu:

- You can choose whether or not the views are organized by type using headers for Sheet Views, Column Views, and Row Views.
- For column and row views only:
  - You can choose whether or not the individual view names display directly on the menu, or whether they are grouped together as part of a group name.
  - If the view names are grouped, you can specify single-selection or multiple-selection of views. On the menu, the impact of this choice determines whether clicking the group name opens a dialog where the user can select multiple views, or whether the view names display as sub-menu items and only a single view can be selected.

This section explains how these options impact the view display on the **Change View** menu of the default Axiom ribbon tab. Keep in mind that it is possible to use the **Change View** command in any custom ribbon tab or task pane and specify different behavior. For example, you can create a button on a ribbon or an item on a task pane that directly activates a particular view (versus showing a menu of all views).

Disabling headings on the menu

By default, the defined views for a sheet display on the **Change View** menu organized by view type. The menu uses headings for **Sheet Views**, **Column Views**, and **Row Views**. If the sheet does not have any views of a certain type, then the heading for that type does not display on the menu.



Example Change View menu with all three view types

If desired, you can disable these headings on a per sheet basis. To do so, change the setting **Show View Headers in Menu** to **Off**. This setting is located in the **Sheet Options > Views** section of the Control Sheet.

Views	
Initial Dynamic View (view on open)	All Detail
Initial Dynamic Column Views (view on open)	Q4
Initial Dynamic Row Views (view on open)	[All]
Current Dynamic View	All Detail
Current Dynamic Column Views	Q3,Q4
Current Dynamic Row Views	Section1,Section2
Show View Headers in Menu	Off

NOTE: When using the Change View command in the task pane to display the view menu, the list of views never has headings, regardless of this setting.

Generally speaking, headers should only be turned off if the sheet only contains one type of view. If the sheet contains multiple types of views, the behavior of the menu may be confusing to end users (when each item on the menu does different things, yet there is no indication that the items are different).

Column view and row view behavior on the menu

The behavior of column views and row views on the **Change View** menu depends on the following factors:

- Whether a group name is defined on the ColumnView tag or RowView tag
- Whether the tag is configured for single-selection

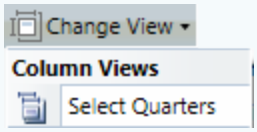
The following examples show some different configurations of ColumnView tags and how they behave on the Change View menu. The same behavior applies to RowView tags.

Example 1

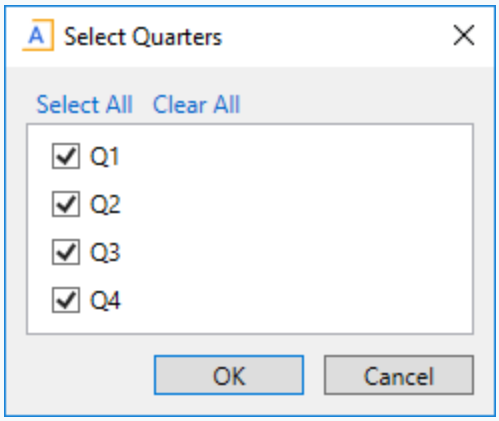
In this example, a group name is defined on the ColumnView tag. The default behavior of multiple-selection applies.

	A	B	C	D	E	F	G	H	I	J
3										
4		[ColumnView; Select Quarters]		[Column;Q1]	[Column;Q1]	[Column;Q1]	[Column;Q2]	[Column;Q2]	[Column;Q2]	[Column;Q3]
5				Jan	Feb	March	April	May	June	July

Because a group name is defined, the view names defined in the individual column tags belong to this group name. On the Change View menu, only the group name is displayed:



When a user clicks on the group name, a selection dialog opens so that the user can choose which column views to make active or inactive. The user can also select or clear all the column views.



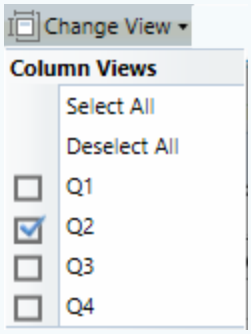
This approach works well with virtually all menu configurations. Additionally, the **Select All** and **Clear All** options in the dialog make it easy for users to toggle all of the views on or off.

Example 2

In this example, the ColumnView tag does not have a defined group name. The default behavior of multiple-selection applies.

	A	B	C	D	E	F	G	H	I	J
3										
4		[ColumnView]		[Column;Q1]	[Column;Q1]	[Column;Q1]	[Column;Q2]	[Column;Q2]	[Column;Q2]	[Column;Q3]
5				Jan	Feb	March	April	May	June	July

Because no group name is defined, the view names defined in the individual column tags display directly on the Change View menu. Users can click directly on these names to make the views active or inactive. They can also choose to **Select All** or **Deselect All**. If a particular view is currently active, the check box displays as checked.



One advantage of this approach is that it is very quick for users to toggle a particular view on or off, versus needing to launch a separate dialog where users must make selections. However, if there are many view options defined in the sheet then the menu may become difficult to use.

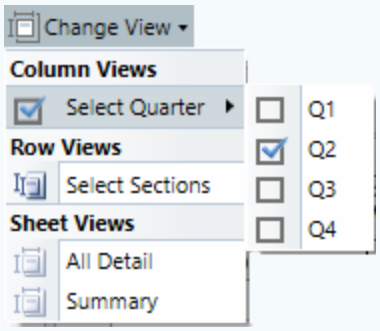
NOTE: If view headers are disabled for the menu, then the select and deselect menu items display as **Select All Columns**, **Select All Rows**, etc.

Example 3

In this example, a group name is defined on the ColumnView tag and single-selection is specified.

	A	B	C	D	E	F	G	H	I	J
3										
4		[ColumnView; Select Quarter;Single]		[Column;Q1]	[Column;Q1]	[Column;Q1]	[Column;Q2]	[Column;Q2]	[Column;Q2]	[Column;Q3]
5				Jan	Feb	March	April	May	June	July

Because a group name is defined, the view names defined in the individual column tags now belong to this group name. But because single-selection is enabled, the individual view names now display as sub-menu items underneath the group name, instead of launching a multiple-selection dialog. When a user clicks on any of the sub-menu items to activate that view name, the other views in this group are made inactive.



When using this approach, it is recommended to define the group name so that it is clear only one item can be selected. For example, use "Select A Quarter" versus "Select Quarters". Keep in mind that if the sheet contains other column views that allow multiple selection, end users may be confused as to why the behavior is different for each view.

Order of views on the menu

If headings are disabled for the **Change View** menu, then all views display on the menu in alphabetical order, regardless of view type.

If headings are enabled for the **Change View** menu (the default behavior), then views display in the order they are found in the sheet (within each individual heading). There is one exception for row and column views: if the view has a group name and single-selection is enabled, then those views display at the top of the list, followed by all other views in the section.

Setting the default views for a sheet

When a file is opened, no views are active by default.

- In the case of sheet views, no sheet views are applied by default so none are active.
- In the case of row and column views, all row and column views are applied by default but they are applied as inactive (meaning in their hidden state).

If you want any views to be active automatically when a file is opened, you can specify those views using the "view on open" settings on the Control Sheet. This is defined on a per sheet basis. There are three settings that control applying views on open, one for each type of view:

Item	Description
Initial Dynamic View	The name of the sheet view to make active when the file is opened. Only one sheet view can be specified.
Initial Dynamic Column Views	The names of the column views to make active when the file is opened. Multiple column view names can be specified, separated by commas. To make all column views active when the file is opened, enter `[All]`. Group names cannot be specified here. If you want to activate all column views in a group, you must list each individual column view in that group (or use `[All]` to activate all column views in all groups).
Initial Dynamic Row Views	The names of the row views to make active when the file is opened. Multiple row view names can be specified, separated by commas. To make all row views active when the file is opened, enter `[All]`. Group names cannot be specified here. If you want to activate all row views in a group, you must list each individual row view in that group (or use `[All]` to activate all row views in all groups).

The initial dynamic view settings are applied the first time the sheet is made active after the file is opened. The view names must exactly match the names defined in the tags, or else the view will not be applied.

NOTE: If you have defined column and row views for a sheet but you do not specify any default column and row views, then all columns and rows that belong to the column and row views will be hidden when the file is opened. If instead you want some or all of these columns and rows to be visible when the file is opened, then you must specify default column and row views.

Example

Views	
Initial Dynamic View (view on open)	All Detail
Initial Dynamic Column Views (view on open)	Q4
Initial Dynamic Row Views (view on open)	[All]
Current Dynamic View	All Detail
Current Dynamic Column Views	Q3,Q4
Current Dynamic Row Views	Section1,Section2
Show View Headers in Menu	On

When this file is opened, the following views will be made active:

- The All Detail sheet view
- The Q4 column view
- All row views, by use of the special keyword `[All]`

For the column views, imagine that this sheet has four column views, one for each quarter. Only the Q4 column view starts off as shown. This means that the columns for the other quarters start off hidden, but the user can go to the **Change View** menu and toggle the other quarters visible as desired.

Also notice the "current dynamic view" settings underneath the "initial dynamic view" settings in the screenshot above. These settings are used by Axiom to track the currently applied views, so that they can be reapplied as needed after certain activities are performed in the file (such as refreshing an Axiom query). The screenshot shows the Control Sheet as it might look after the user has opened the file and then changed some of the view selections. For example, in the **Current Dynamic Column Views** field, you can see that the user has toggled the Q3 column view visible in addition to the Q4 column view that was activated when the file was opened.

Defining GoTo bookmarks for sheet navigation

Using the GoTo feature, you can define bookmarks in Axiom files. When users are in a file, they can use the GoTo menu item to quickly select and go to these defined bookmarks.

GoTo bookmarks can be defined on any sheet in the workbook. When users select the GoTo menu item, they see a list of all bookmarks defined in the workbook, grouped by GoTo column names.

To define GoTo bookmarks in a sheet, you must define a GoTo column in that sheet, and then define bookmark spots in that column.

GoTo tag summary

Tag Type	Tag Syntax
Primary tag	`[GoToColumn;HeaderText;AlphaSort]`
Row tags	`[GoTo;BookmarkText]`

Defining a GoTo column

A GoTo column serves two purposes:

- It signals to Axiom that you want to use GoTo functionality on the sheet.
- It contains the individual bookmarks.

To define a GoTo column, place the following tag in any column in the sheet, within the first 500 rows:

`[GoToColumn;HeaderText;AlphaSort]`

The primary tag uses the following parameters:

Parameter	Description
HeaderText	The text to appear in the GoTo menu for this set of GoTo bookmarks. The header text might be the name of the sheet, or the type of data that the individual bookmarks go to.
AlphaSort	Optional. Specifies whether the GoTo bookmarks are sorted in alphabetical order. • If `False` or omitted, the GoTo bookmarks are listed in the order they appear in the sheet, from top to bottom. • If `True`, the GoTo bookmarks are listed in alphabetical order.

You can have multiple GoTo columns defined in a sheet.

NOTES:

- The primary GoToColumn tag must be located in the first 500 rows of the sheet.
- GoToColumn tags can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Defining GoTo bookmarks

The GoTo bookmarks are the specific spots in the sheet that you want a user to be able to quickly jump to. When a user selects a bookmark, they will be taken to the front of the tagged row, honoring any freeze panes settings.

To define a GoTo bookmark, place the following tag in a GoTo column, within the row that you want users to jump to:

`[GoTo;BookmarkText]`

BookmarkText is the text to appear in the GoTo menu, underneath the applicable GoTo header text. The text should describe the contents of the bookmark. For example: `[GoTo;Budget Summary]`. When a user selects the GoTo menu item, they would select the "Budget" header and then select the bookmark "Budget Summary" to go to that point in the Budget sheet.

A GoTo column can contain any number of GoTo bookmarks.

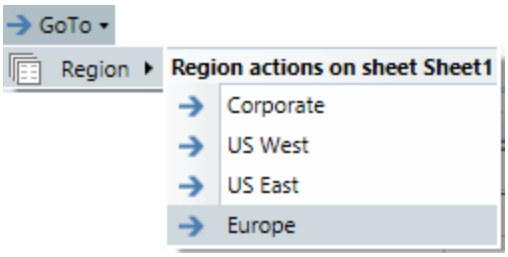
NOTE: The row tag can be placed within a formula if desired. For example, you might want to construct the Bookmark Text dynamically, using a cell reference to point to header text in the section.

GoTo example

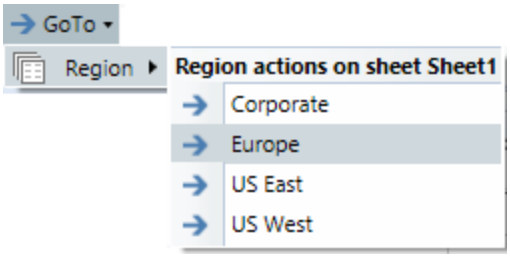
The following screenshot shows an example GoToColumn tag with several GoTo bookmarks defined. This is not a real implementation example; the intent is simply to show the tags. In a real example each GoTo tag might mark the start of a new section in a report, so that users could jump directly to the data for a specific region.

	A	B
1		
2		
3		[GoToColumn; Region]
4		
5		
6		
7		
8		[GoTo; Corporate]
9		
10		
11		[GoTo; US West]
12		
13		
14		[GoTo; US East]
15		
16		
17		[GoTo; Europe]

The GoTo menu would list the bookmarks in the order they are found in the sheet:



Alternatively the primary tag could specify alphabetical sorting like so: `[GoToColumn;Region;True]`. In this case the GoTo menu would list the bookmarks in alphabetical order:



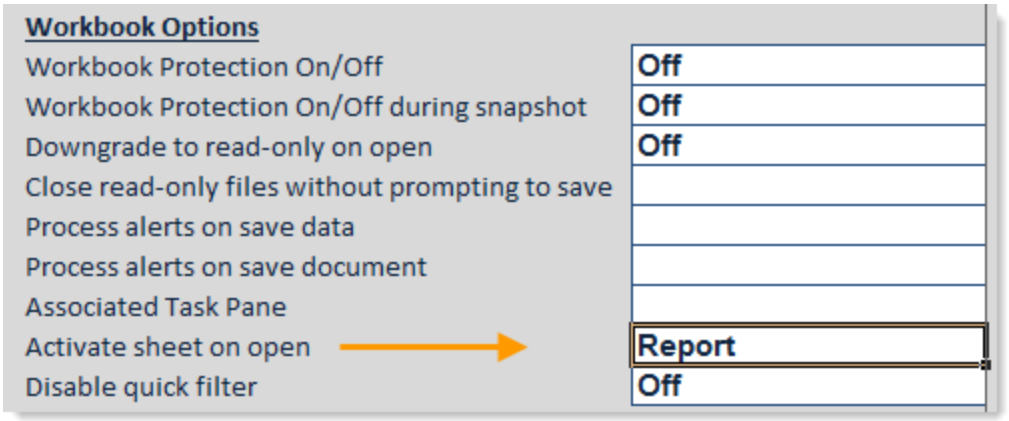
Setting the active sheet and active cell

You can specify the active sheet for an Axiom file, so that the file will always open to the designated sheet. Additionally, for each sheet you can specify the active cell, so that the cursor will always start in the same place when the file is opened.

Both of these settings can only be made on the Control Sheet.

Setting the active sheet

To set the active sheet for a file, use the **Activate sheet on open** option, located in the **Workbook Options** section of the Control Sheet. Enter the name of any visible sheet in the file.



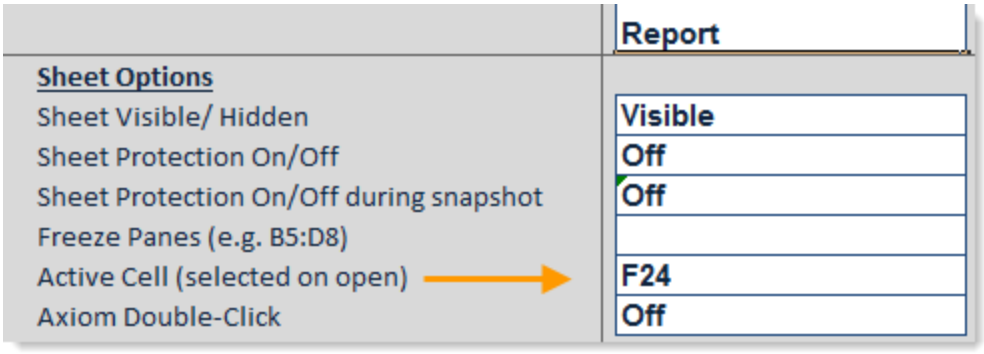
When the file is opened, the sheet designated here will be the active sheet, regardless of which sheet was active when the file was last saved. This option is useful to keep the active sheet consistent for end users, rather than accidentally set it to whatever sheet the file designer is working on.

If this setting is left blank, then the file will open to whichever sheet was active when the file was last saved.

NOTE: If special features are used to open the file to a specific sheet—such as the GetDocument function, or an alert notification—then that designated sheet takes precedence over the active sheet defined in the Control Sheet.

Setting the active cell

To set the active cell for a sheet, use the **Active Cell** option, located in the **Sheet Options** section of the Control Sheet. Enter the desired cell address, such as F24.



When the file is opened, the specified cell is made active on that sheet. This means that the cursor is placed in this cell, and the sheet is re-oriented so that this cell is positioned in the top left-hand corner (after honoring any freeze panes settings).

This option is useful for any file where you want the user to start at a certain place in the sheet, and you do not want the active cell to be inadvertently left on whichever cell was active when the file was last saved.

NOTES:

- If the sheet has an Initial Dynamic View assigned and that view has a specified active cell, then the active cell for the view takes precedence over the active cell defined in the Control Sheet.
- If special features are used to open the file to a specific cell—such as the GetDocument function, or an alert notification—then that designated cell takes precedence over the active cell defined in the Control Sheet.

Configuring default display options for a sheet

You can configure various display options for sheets in Axiom files on the Control Sheet. These settings are applied when a user opens the Axiom file and the sheet is made active.

It is often useful to set these values in the Control Sheet instead of using Excel features, so that the desired initial state of the file is not inadvertently changed when saving the file. For example, you might want the zoom level of a particular sheet to always start at 85%. You can set this zoom level in Excel and save the file, but this setting may be inadvertently lost if another user later changes the zoom level to 120% and saves the file. Instead, you can set the zoom level to 85% on the Control Sheet, and this setting will always be applied, regardless of the zoom level when the file was last saved.

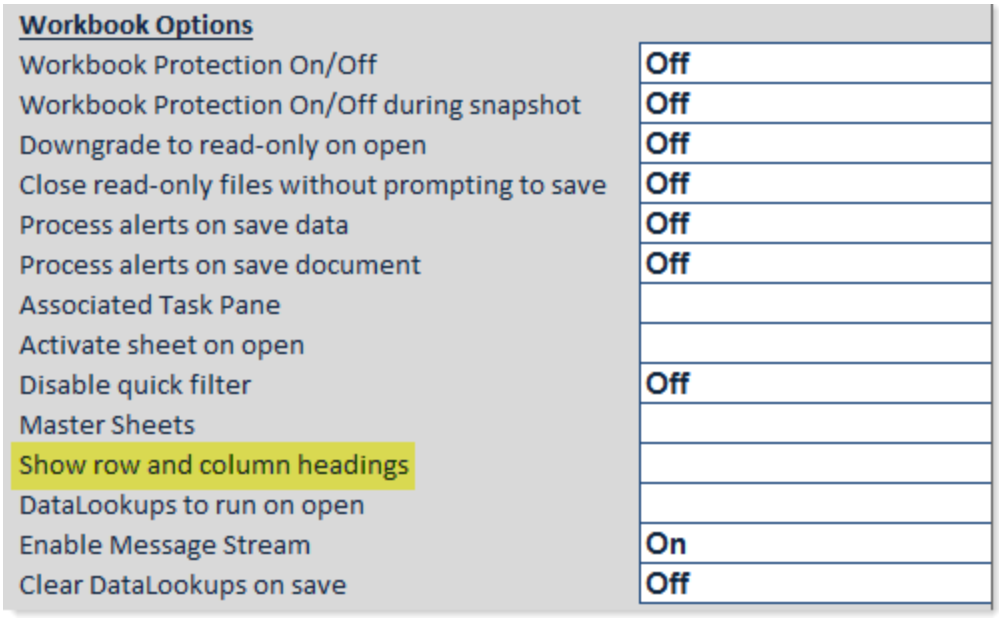
Headings

You can specify whether the spreadsheet row and column headings are initially visible when a user opens an Axiom file. Row and column headings are often hidden to reduce visual clutter when displaying sheets to end users.

NOTE: This option is set at the workbook level and applies to all sheets in the file.

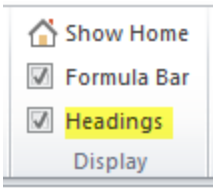
To set the visibility of row and column headings, use the **Show row and column headings** property in the **Workbook Options** section of the Control Sheet. This property can be set as follows:

- **<Blank>**: Row and column headings are visible if the file is opened read/write, and hidden if the file is opened read-only. This is the default setting.
- **On**: Row and column headings are visible when the file is opened.
- **Off**: Row and column headings are hidden when the file is opened.



Workbook Options	
Workbook Protection On/Off	Off
Workbook Protection On/Off during snapshot	Off
Downgrade to read-only on open	Off
Close read-only files without prompting to save	Off
Process alerts on save data	Off
Process alerts on save document	Off
Associated Task Pane	
Activate sheet on open	
Disable quick filter	Off
Master Sheets	
Show row and column headings	
DataLookups to run on open	
Enable Message Stream	On
Clear DataLookups on save	Off

This only determines the initial state of the row and column headings. Once the file is opened, users can choose to toggle the headings shown or hidden for the current session by using the **Headings** check box in the **Display** group of the **Axiom** ribbon tab (or by using Excel functionality).



Additionally, clicking **Show Everything** on the Axiom Designer tab will show the headings.

Freeze panes

You can define the freeze panes area for each sheet in an Axiom file. The freeze panes feature is useful to "hide" setup areas at the top and left of the sheet, and to lock header rows and columns in place for scrolling purposes.

To define the freeze panes area of a sheet, use the **Freeze Panes** property in the **Sheet Options** section of the Control Sheet. Enter the top left and bottom right cells of the area you want frozen—for example: `B5:D8`. B5 indicates the start of the frozen area, and D8 indicates the first unfrozen cell. Columns B and C will be frozen, and rows 5:7 will be frozen.

Sheet Options	
Sheet Visible/ Hidden	Visible
Sheet Protection On/Off	Off
Sheet Protection On/Off during snapshot	Off
Freeze Panes (e.g. B5:D8)	B5:D8
Active Cell (selected on open)	D8
Axiom Double-Click	Off
Enable Drilling	On
Master Sheet	
Show Gridlines	Off
Default Zoom	90%

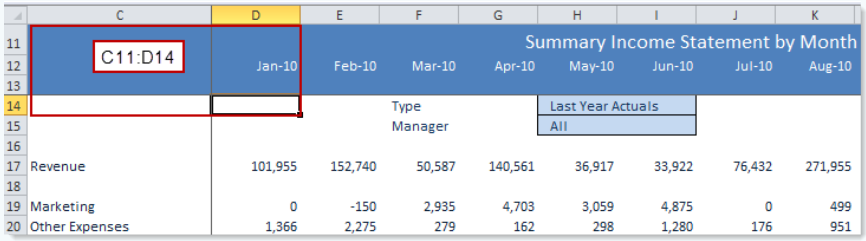
The freeze panes setting is applied when the file is opened. Users can only unfreeze panes if the sheet is not protected, or if they have security rights to unprotect the sheet. You can toggle the freeze panes on and off from **File Options > Freeze Panes**.

Freeze panes settings are reapplied when a view is applied. This means that if a view has been applied to the sheet (either manually by a user, or automatically via the **Initial Dynamic View** setting), then freeze panes will be reapplied each time the sheet is refreshed, since the view is automatically reapplied after refresh. If desired, you can use a formula to determine what the freeze panes settings should be, based on the current view.

NOTE: If a file has been saved with applied freeze panes, then clearing the **Freeze Panes** setting will not remove the frozen panes. If you want to remove the frozen panes entirely, you must clear the setting on the Control Sheet, and then manually toggle the frozen panes off, and then save the file.

Example

In the following report, the freeze panes setting is C11:D14:



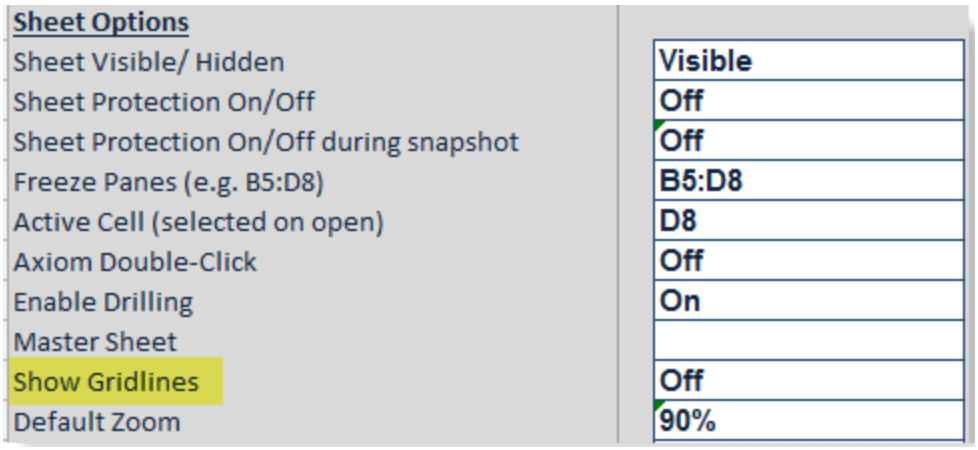
The frozen panes start at the first cell in the range (C11) and then continue to the top and left of the last cell in the range (D14). When you scroll across, column C is always frozen onscreen. When you scroll down, rows 11-13 are always frozen onscreen.

Gridlines

You can specify whether gridlines should be visible by default for each sheet in an Axiom file. Gridlines are often hidden to reduce visual clutter when displaying sheets to end users.

To set the visibility of gridlines, use the **Show Gridlines** property in the **Sheet Options** section of the Control Sheet. This property can be set as follows:

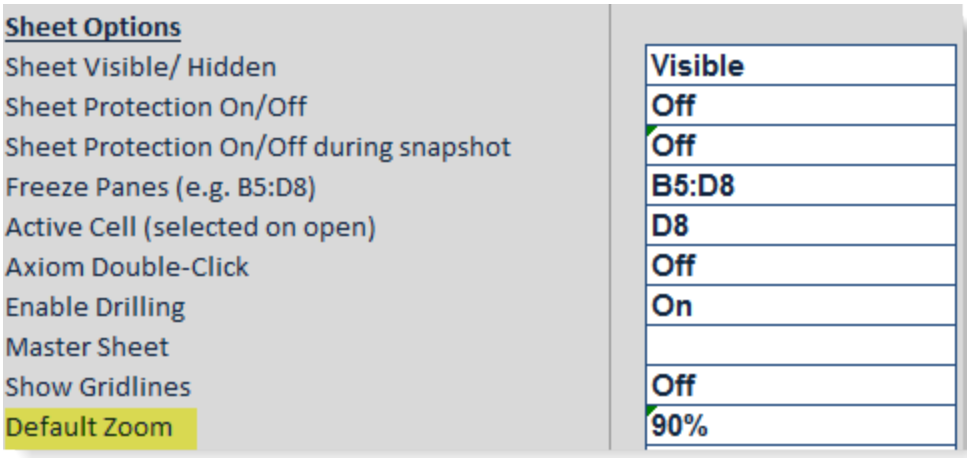
- **<Blank>**: Gridlines are left as is when the file is opened. The visibility of gridlines is controlled by Excel functionality.
- **On**: Gridlines are visible when the file is opened.
- **Off**: Gridlines are hidden when the file is opened.



Once the file is opened, the user can toggle the gridlines visible or hidden using Excel functionality, for the current session. The next time the file is opened, the Control Sheet setting is applied again.

Zoom

You can specify the default zoom level for each sheet in an Axiom file. To set the zoom level, use the **Default Zoom** property in the **Sheet Options** section of the Control Sheet. Enter a percentage as an integer, with or without a percentage sign—for example, either `90` or `90%` is valid.



Sheet Options	
Sheet Visible/ Hidden	Visible
Sheet Protection On/Off	Off
Sheet Protection On/Off during snapshot	Off
Freeze Panes (e.g. B5:D8)	B5:D8
Active Cell (selected on open)	D8
Axiom Double-Click	Off
Enable Drilling	On
Master Sheet	
Show Gridlines	Off
Default Zoom	90%

When the file is opened, the default zoom level is applied to the sheet. The user can change the zoom level using Excel functionality, for the current session. The next time the file is opened, the Control Sheet setting is applied again.

If Default Zoom is blank, then the zoom level in the sheet is left as is. The zoom level is controlled by Excel functionality.

Configuring sheet visibility for Axiom files

Each sheet in an Axiom file can be visible or hidden when the file is opened.

To configure sheet visibility settings, use the **Sheet Visible/Hidden** setting in the **Sheet Options** section of the Control Sheet. This setting takes the following values:

- **Visible** – The sheet is visible to all users.
- **Hidden** – The sheet is hidden by default.

Whether users can unhide a hidden sheet depends on whether the file also has workbook protection enabled:

- If workbook protection is enabled for the file, then hidden sheets cannot be unhidden using Axiom functionality unless the user is an administrator, or unless the user has the global **Remove Protection** permission or the file-specific **Allow Unprotect** permission. Plan files always have

workbook protection enabled; for other files workbook protection is optional depending on the Control Sheet setting.

- If workbook protection is not enabled for the file, then users can unhide hidden sheets using normal spreadsheet functionality.

Sheet visibility settings are applied when the file is opened. If you change the setting on the Control Sheet, the new setting will be applied the next time the file is opened. If you manually unhide a hidden sheet, the sheet will be hidden again the next time the file is opened.

NOTE: The ability to hide a sheet should not be used as a substitute for security. Hidden sheets are useful to hide setup information that end users do not need to see, but they should not contain data that end users could not otherwise access.

Using very hidden

Axiom supports a third visibility option of "very hidden." If a sheet is very hidden, then you cannot use Microsoft Excel functionality to unhide it. The only way to unhide a very hidden sheet is from within the VBA module (or to change the sheet visibility settings on the Control Sheet).

The "very hidden" option is not available from the drop-down list on the Control Sheet; you must manually enter it into the cell. To do so:

- Unprotect the Control Sheet, and then remove the data validation from the cell using standard spreadsheet features.
- Type `veryhidden` (with no space) into the cell.

Hiding control sheets

The default Control Sheet has special behavior to control whether it is visible or hidden by default. For more information, see Viewing the Control Sheet.

Other control sheets are automatically hidden or visible depending on whether the user has the security permission for the corresponding feature. You do not need to manually hide these control sheets in order to hide them from end users. The hide behavior is the same as described earlier for the **Sheet Visible/Hidden** setting.

The following table summarizes the security permissions necessary for a control sheet to be visible in a file. If a user does not have this permission, the sheet is hidden automatically.

Control Sheet	Security Permission
Alerts	Allow Unprotect and Read/Write access
Batch	Allow File Processing
Collect	Allow File Processing

Control Sheet	Security Permission
Drilling	Allow Sheet Assistant
File Processing	Allow File Processing
Form	Allow Sheet Assistant

NOTE: If you have manually hidden a control sheet in a file, then this behavior will not apply because Axiom cannot override the manual hide. It is not recommended to manually hide any sheets in Axiom files.

Configuring worksheet protection for Axiom files

You can specify whether worksheet protection is applied to each sheet in an Axiom file. To edit the protection settings for a sheet, use the **Sheet Protection On/Off** option in the **Sheet Options** area of the Control Sheet. (This setting is also available via the Sheet Assistant.)

- If **Sheet Protection** is set to **On**, then sheet protection is applied to the sheet when the file is opened. Generally, users are restricted to editing unlocked cells. In plan files, users can also use the calc method library to insert new lines or change existing lines.
- If **Sheet Protection** is set to **Off**, then sheet protection is not applied to the sheet by Axiom. However, any existing sheet protection is not removed.

When sheet protection is enabled, by default Axiom applies a system password to the sheet. This is intended to block casual users from using standard Microsoft Excel functionality to unprotect the sheet. Only administrators and users with the global **Remove Protection** permission or the file-specific **Allow Unprotect** permission can unprotect the sheet using Axiom functionality.

IMPORTANT: Sheet protection for Axiom files should always be applied via the Control Sheet, not by manually using Excel's spreadsheet features. Manually applied protection—especially if used with custom defined passwords—will not work with all Axiom features. Additionally, manually applied protection will prevent the file from opening in the Windows Client.

NOTES:

- If a file has sheet protection enabled on the Control Sheet, and then later you change the setting to **Off**, the sheet protection will not be removed from the file until you remove it using the Axiom unprotect feature (**Advanced > Protect > Worksheet**) and then save the file. The next time the file is opened, the protection will not be reapplied.
- If you want worksheet protection to be applied when a snapshot copy is taken, remember to also set **Sheet Protection On/Off during snapshot** to On.

Sheet protection settings

The sheet protection applied by Axiom allows users to do the following:

- Select locked cells
- Select unlocked cells
- Format cells

  The behavior of this setting differs between the Excel Client and the Windows Client. In the Excel Client, this allows any cell in the protected sheet to be formatted. In the Windows Client, only unprotected cells can be formatted.

- Use AutoFilter

  This setting only applies to the Excel Client. AutoFilter is not useable in protected sheets in the Windows Client.

It is not possible to customize the Axiom protection settings to allow users to perform other actions or to disallow the default actions.

When designing templates with a comment column, make sure to set the size of the comment column to a reasonable width, because end users will be unable to resize the column due to the sheet protection settings.

When adjusting the protection settings for a cell, keep in mind that enabling the Hidden property for a cell may affect certain Axiom functionality. For example, if a Print tag is placed in a cell that is Hidden, Axiom will not detect the print view.

Disabling password protection for worksheets

If desired, you can disable password protection for worksheets by editing the system configuration setting **ExcelWorksheetPasswordProtectionEnabled**. Eliminating the password can significantly improve file performance when using Axiom files in Excel. Some clients may decide that the password protection is not necessary in their environment and they would rather benefit from the improved performance.

By default this setting is enabled, which means that when Axiom applies worksheet protection per the Control Sheet settings, the protection includes a password. Users who have the appropriate Axiom security permissions can remove this protection without entering the password—the system automatically applies the password when using the Axiom protection removal features. However, any user who attempts to manually remove the protection using Excel's protection removal features will be prompted to enter the password. This password is intended to discourage casual attempts to breach the worksheet protection.

If this setting is disabled, then the worksheet protection is still applied by Axiom but the protection does not include a password. This means that any user with the knowledge to use Excel's protection removal features can remove the worksheet protection.

IMPORTANT: All clients should keep in mind that workbook and worksheet protection are not substitutes for security. Workbook and worksheet protection are useful for many things—such as hiding setup information in a file and focusing end user input and attention to designated areas—but these protections are not meant to *prevent* user access to protected areas, they are only meant to *discourage* access. If a workbook contains *any* data that a particular user should not be able to see, then that user should not have security permission to access that workbook.

Configuring workbook protection for Axiom files

Workbook protection is configured differently for different types of files.

- **Templates / plan files:** Workbook protection is always applied to plan files, regardless of the Control Sheet setting. However, templates are not protected so that template authors can configure the workbook as needed.
- **All other Axiom files:** You can configure a file to be protected by enabling the **Workbook Protection On/Off** setting on the Control Sheet. The protection is applied when the file is opened.

NOTE: If you want protection to be applied when a snapshot copy is taken of the file, remember to also set **Workbook Protection On/Off during snapshot** to On. This applies to all files, including plan files.

Workbook protection is applied using a system password. This is intended to block casual users from unhiding sheets, adding or removing sheets, or renaming sheets using standard Microsoft Excel functionality. Only administrators and users with the global **Remove Protection** permission or the file-specific **Allow Unprotect** permission can unprotect the workbook using Axiom functionality.

If a user unprotects the workbook during a session, the protection settings are reapplied the next time the file is opened.

Microsoft Excel's workbook protection feature allows protection for structure and windows. Axiom files use the default **Structure** protection only.

IMPORTANT: All clients should keep in mind that workbook and worksheet protection are not substitutes for security. Workbook and worksheet protection are useful for many things—such as hiding setup information in a file and focusing end user input and attention to designated areas—but these protections are not meant to *prevent* user access to protected areas, they are only meant to *discourage* access. If a workbook contains *any* data that a particular user should not be able to see, then that user should not have security permission to access that workbook.

Axiom Queries

Axiom queries can be used to update and/or insert data into a sheet. Using an Axiom query, you define:

- The set of data to be queried from the database, by completing data query settings on the Control Sheet
- The placement of that data on the target sheet, by placing database column names on a reserved row (or column) in the sheet, and by defining data ranges where records will be inserted and/or updated
- The formatting and any additional formulas to be applied to each inserted record of data, by using the calc method library or an in-sheet calc method
- How data should be updated when data refreshes are performed, by completing the update settings on the Control Sheet

Axiom queries can be used in any Axiom file, but are most likely to be used in templates/plan files and report files. In templates, Axiom queries are typically used to dynamically populate the plan file with the data relevant to each individual plan code. In reports, Axiom queries can be used for any data query. While Axiom queries are well-suited for data that needs to adjust dynamically, they can also be used for configurations that are static in nature (update-only queries).

About Axiom queries

Axiom queries can be used to bring data, formatting, and calculations into Axiom files, or to update existing data rows. Axiom queries are set up using the following components:

- Axiom query settings defined on the file's Control Sheet.
- Various control rows and columns defined in the target sheet. These control rows and columns specify the data to be brought into the sheet and where, and in some cases also define the formatting and formulas to be applied (in-sheet calc methods).
- Calc methods defined in a calc method library for the target sheet. This only applies when setting up Axiom queries for templates/plan files.

Any sheet can have an Axiom query, and each sheet can have multiple Axiom queries.

Vertical and horizontal queries

An Axiom query can be oriented vertically or horizontally.

- **Vertical queries**: Vertical queries bring records into the sheet as rows. This is the "standard" type of Axiom query that is used by default on the Control Sheet. All queries are vertical queries unless you explicitly flag the query as a horizontal query.
- **Horizontal queries**: Horizontal queries bring records into the sheet as columns. Horizontal queries can be used on their own for certain specialized data configurations, or as a "building block" for a second, vertical Axiom query. For example, you can use a horizontal query to dynamically build out the field definitions for a vertical Axiom query. Horizontal queries generally require more advanced report knowledge to set up.

Most Axiom query settings apply to either type of query. However, some settings can only be used with standard vertical queries, such as spreadsheet grouping and calc method libraries.

The orientation of the query determines how the control rows and columns must be placed in a sheet. For example, the field definition for a query contains the database column names for the data to be brought into the sheet. If the query is vertical, then the field definitions are placed in a designated row, so that the associated data can be brought in as rows. If the query is horizontal, then the field definitions are placed in a designated column, so that the associated data can be brought in as columns.

Axiom query settings on the Control Sheet

The settings on the Control Sheet define the following for an Axiom query:

- Data query settings, such as the primary table to query, what level to sum rows by, and whether to apply a data filter
- The location of various control rows and control columns in the target sheet
- Data refresh and insertion options
- Data sorting options
- Calc method options

For more information, see Axiom query settings.

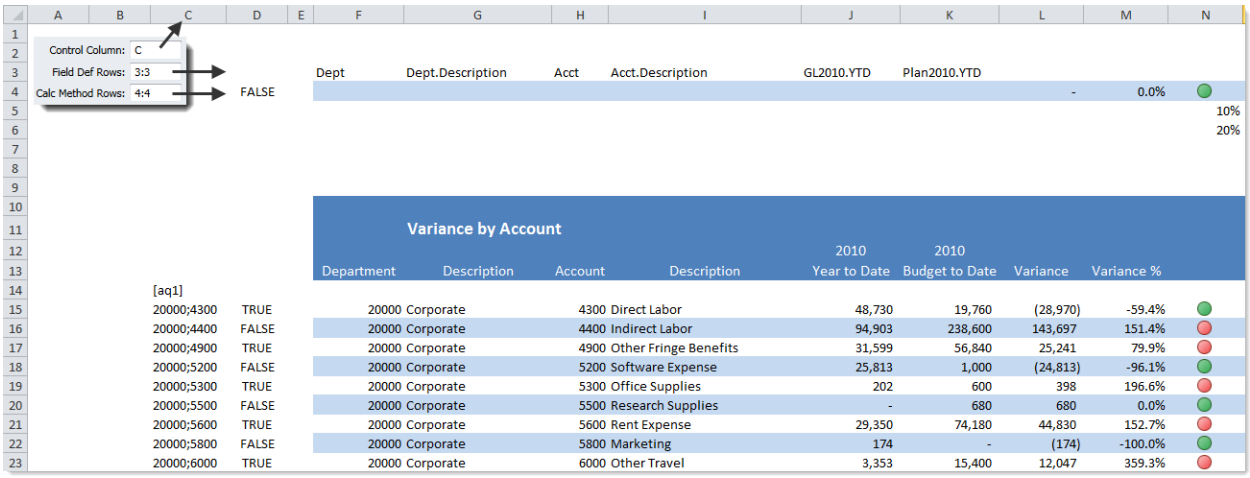
Axiom query settings can be defined using either the Sheet Assistant or the Control Sheet. Some advanced query settings are only available on the Control Sheet.

NOTE: When an Axiom query is processed, the user's table filters (as defined in Security) are always applied to the query, in addition to any data filter defined for the query. This means that the data returned by an Axiom query may vary, depending on the user processing the query.

Control rows and columns for Axiom queries

The following control rows and columns must be defined within the target sheet. Whether each item is a row or a column depends on whether the query is a vertical query or a horizontal query.

Control Row or Column	Description	More Information
Field definition	This item contains the database column names for the data to be brought into the target sheet. The column names are placed in the columns or rows where you want the data to be populated. Vertical queries use field definition rows, and horizontal queries use field definition columns. The field definition for a query can use multiple rows or columns.	Creating the field definition for an Axiom query
Insert control	This item contains the keywords to specify the beginning and end of a data range on the target sheet. You can have multiple data ranges for each Axiom query, using different filters to create sub-groups of data. Vertical queries use an insert control column, and horizontal queries use an insert control row.	Creating the insert control column or row for an Axiom query
In-sheet calc method	This item contains the formatting and formulas that you want to apply to records of data as they are inserted. Vertical queries use in-sheet control rows, and horizontal queries use in-sheet control columns. The in-sheet calc method for a query can use multiple rows or columns. For plan files, in-sheet calc methods are optional. Plan files can use the calc method library instead.	Using in-sheet calc methods with Axiom queries
Data control	This item is used by Axiom to record the key codes for each record of data inserted into the worksheet. These key codes can be used to perform updates of existing data. Vertical queries use a data control column, and horizontal queries use a data control row. By default, the data control column/row is the same as the insert control column/row.	About data control codes for Axiom queries



Example Axiom query control rows and control columns (vertical query)

How Axiom queries are run

If an Axiom query is active in an Axiom spreadsheet file, the query can be run to bring data into the file. The methods used to run a query depend on the type of file and on user permissions. The following features represent the primary means of running Axiom queries.

Feature	Description
Refresh	The **Refresh** feature in the **File Options** group can be used in any Axiom file to run all active Axiom queries in the file, or for the current sheet only. This feature is most typically used in reports, to refresh the file with the most current data. In plan files, **Refresh** is only available to administrators and users with the **Run AQs in Plan Files** security permission. This is to prevent users from accidentally overwriting data in plan files. In most cases, Axiom queries are only run in plan files by administrators using the **Process Plan Files** utility. If an Axiom query needs to be run in plan files, it can be configured to automatically refresh when the file is opened. Administrators also have access to the **Axiom Designer** tab, which provides additional refresh options such as the ability to refresh a single Axiom query at a time. See Using the Axiom Designer tab.

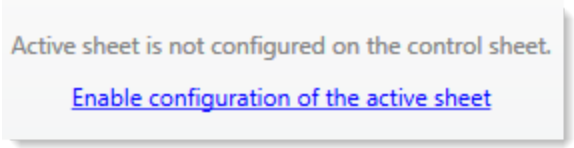
Feature	Description
Process Plan Files	The **Process Plan Files** utility can be used to selectively process Axiom queries in plan files. Only active queries can be selected. This utility is typically used to bring data into plan files when they are initially created. If your plan files are designed to accommodate periodic data updates, this utility can be used to update plan files in batch. From the file groups menu, this utility is only available to administrators and users with the **Process Plan Files** security permission. The utility can also be run using Scheduler.
File Processing	The File Processing feature refreshes the file, including running Axiom queries, and performs a file processing action such as taking a snapshot of the file or creating an export file. If a file is set up to use file processing, then any user with access to the file can execute the file processing. File processing can also be run using Scheduler.
Process Document List	This Scheduler task can be used to process all active Axiom queries in any Axiom file. Its primary intent is to process driver files or report utilities.

Axiom queries can also be configured to automatically run when a file is opened, or after a save-to-database has occurred, or to support various specialty features.

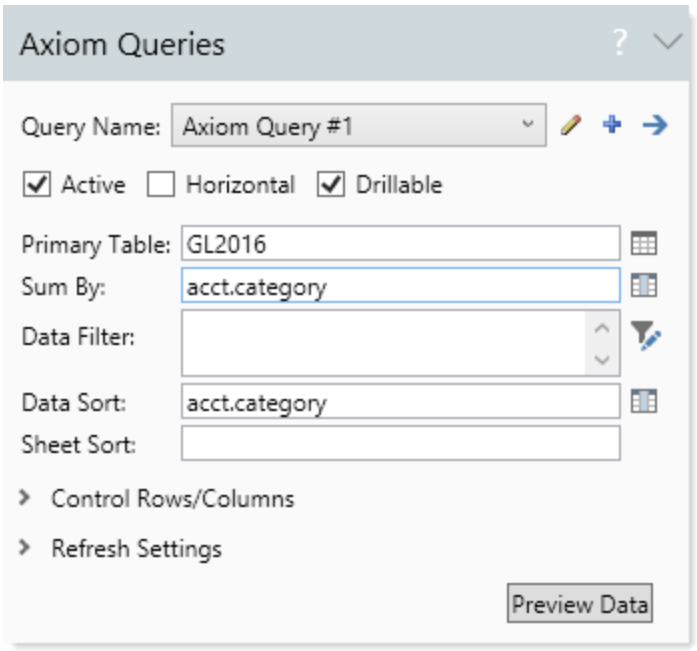
Defining Axiom queries

The basic process steps for defining Axiom queries are as follows:

1. If the target sheet is not already set up on the Control Sheet, enable it. You can do this by typing the sheet name into the first available column on the Control Sheet, or you can use the Sheet Assistant to automatically enable the sheet.



2. Enable the Axiom query (select **Active**), and then complete the Axiom query settings as desired. You can modify the Control Sheet directly, or use the Sheet Assistant.



For more information, see:

- Axiom query settings
- Constructing the database query for an Axiom query
- Refresh and insertion behavior for Axiom queries

3. Set up the control rows and columns on the target sheet, in the locations specified in the query settings:
   - Field definition – see Creating the field definition for an Axiom query.
   - Insert control – see Creating the insert control column or row for an Axiom query.
   - In-sheet calc method (required for reports, optional for plan files) – see Using in-sheet calc methods with Axiom queries.
4. If you are defining an Axiom query in a template, and you want to use the calc method library, complete the calc method settings as appropriate. See Using calc method libraries with Axiom queries.

   NOTE: Calc method library settings are not available in the Sheet Assistant. If you want to use a calc method library instead of an in-sheet calc method, you must modify the Control Sheet directly.

5. Test the Axiom query to make sure it is bringing in data as desired.

As a preliminary test, you can use the **Preview Axiom Query Data** button on the Sheet Assistant. This shows you the raw data that the query returns from the database. This may be useful to help ensure that your data query settings are set up as intended. For example, if the query returns 200 records of data when you are only expecting 50, then this may mean that the sum level of the query needs to be adjusted, or that the data filter is not set as intended.

To test all aspects of the query, you can refresh the file to run the query. If you are defining the Axiom query in a template, the best way to test the query is to build a plan file from the template and then test in the plan file.

Adding Axiom queries to a Control Sheet

By default, the Control Sheet contains settings for two Axiom queries. If you need more Axiom queries, you can use the Sheet Assistant to add them:

1. In the Axiom Assistant area, select the **Sheet Assistant** tab (if it is not already the active tab).
2. If the Control Sheet is currently the active sheet, move to the sheet where you want to add an Axiom query. The Axiom Query options do not display in the Sheet Assistant if you are currently on the Control Sheet.
3. In the **Axiom Queries** section of the Sheet Assistant, click the **Add Axiom Query** button + to the right of the **Axiom Query** list.

A new Axiom query section is added to the Control Sheet, underneath the last defined section. You can now define the settings for the new query as appropriate, using either the Sheet Assistant or the Control Sheet.

Constructing the database query for an Axiom query

The database query settings of an Axiom query define the set of data to be available to bring into the target sheet. Once the data has been queried from the database, Axiom looks to the *field definition row* (or column) and the *insert control column* (or row) to determine exactly what data to bring into the sheet and where.

This section provides additional details about the database query settings, to help you obtain the desired results.

Specifying the primary table for an Axiom query

The **Primary Table** setting defines the default table for the Axiom query. In most cases, the primary table is a data table, but it does not have to be. For example, the primary table could be a reference table such as ACCT, if you wanted to generate a list of all current accounts.

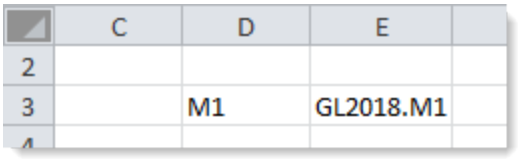
The primary table affects which tables and columns can be used in other Axiom query settings. All settings must be valid in the context of the primary table. Generally speaking, you can use columns from tables other than the primary table as follows:

- If the primary table is a data table, then you can also query data from any lookup reference tables, and from additional data tables that share at least one column with the data table.
- If the primary table is a reference table, then you can also query data from any lookup reference tables, and from any tables with a direct lookup to the primary table.

Columns from other tables must be specified using full Table.Column syntax. For more information, see Using multiple tables in an Axiom query.

Applying the primary table to column-only settings in the Axiom query

If any settings in the Axiom query use only a column name (instead of a fully qualified Table.Column name), then the primary table will be assumed as the table for that column. For example, if you want to bring in data from the M1 column of the GL2018 table, then you can enter either of the following into the field definition:



	C	D	E	
2				
3		M1	GL2018.M1	
4				

In this example, if the primary table is GL2018 then both entries will return the same values, because the M1 entry is interpreted as `PrimaryTable.M1` (`GL2018.M1`). But if the primary table is GL2017, then the M1 entry will be interpreted as `GL2017.M1` instead.

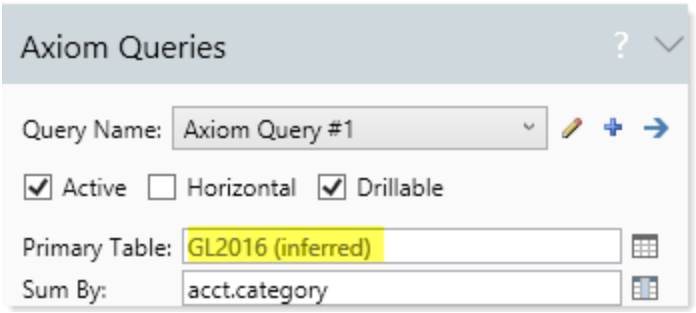
However, if a column-only entry is a validated column (such as ACCT), then Axiom assumes the lookup table instead of the primary table. For example, an entry of ACCT will be interpreted as `ACCT.ACCT` instead of `GL2018.ACCT`. This is done so that the query can support pulling in data from multiple data tables that all use ACCT as a lookup table.

Omitting the primary table

If you are using alias names in the field definition, you can omit the primary table setting, and Axiom will assume the primary table based on the entries in the field definition. This allows you to change the table that the alias names point to, without needing to update the primary table setting on the report.

When the primary table is omitted, Axiom assumes the primary table based on the first data column it finds in the field definition (moving from left to right for vertical queries, and top to bottom for horizontal queries). Reference table columns are ignored. The data column must be an alias name or a fully qualified Table.Column name in order to assume the primary table.

If you want to see which table is being assumed as the primary table for an Axiom query, check the Sheet Assistant. When the primary table is not explicitly defined, the assumed table name displays here in gray font and with the text **(inferred)** after the table name.



Axiom will only assume a reference table as the primary table if no data columns can be identified in the field definition (either no data columns exist, or they cannot be associated with a table).

Primary table design considerations

In many cases, you can achieve the same basic query results using different primary tables. For example, if you are reporting on data from GL2020 and BGT2020, and setting the sum by to the Dept lookup column, then the primary table can be any of the following:

- GL2020
- BGT2020
- Any other data table with compatible dimensions (GL2019, BGT2021, and so on)
- Dept

However, the primary table affects the tables and columns that you can use in other Axiom query settings, and the features available to the query. The following design considerations can impact what you choose to set as the primary table:

- Which table columns do you want to include in the field definition? For example, if a data table has lookups to the Dept and Acct reference tables, then when the data table is the primary table you can include columns from either or both of the Dept or Acct reference tables. But if the primary table is Dept then you cannot include columns from the Acct table (unless the Dept table has a lookup to it, which is not likely).

- Which table columns do you want to use in the sum by, sort, or filter? For example, you might want to sum by Dept but filter the query by Acct. That is possible if the primary table is a data table, assuming the data table looks up to both reference tables. But if the primary table is Dept then you cannot filter by the Acct table (unless the Dept table has a lookup to it, which is not likely). Other, more advanced query settings are subject to the same basic considerations.
- When the sum by is a lookup column, do you want to return all records in the lookup table, or just the records for which you have data? For example, if the sum by is Dept and the primary table is the Dept reference table, then the query will return all departments in that table. But if the sum by is Dept and the primary table is the GL2020 data table, then the query will only return the departments used in GL2020. (If you need to return all departments but use a data table as the primary table, then you must use a nested query setup where the first query brings in all departments from the Dept table, and then the second query brings in the data from the data table.)
- Do you need to be able to drill the report, and if so what dimensions do you want to drill by? If the primary table is a data table, then you can drill using any dimensions that the primary table looks up to. If the primary table is a reference table, then you can only drill by that dimension (or a "child" dimension that the reference table looks up to).

Using multiple tables in an Axiom query

Most Axiom queries use more than one table in the query settings. For example, the primary table is typically a data table, such as GL2020, but the data table links to one or more reference tables, such as ACCT or DEPT. Additionally, you may want to include data from multiple data tables—for example, to compare actual data to plan data.

It is important to structure your Axiom query appropriately so that the data results are as expected.

▶ Including reference tables when the primary table is a data table

Axiom queries can include data from any reference table that the primary table has a lookup relationship with. By default, when a reference table is included in an Axiom query, it is joined to the data table by using an *inner join*.

For example, if the reference table column ACCT.ACCT is the assigned lookup column for the data table column GL2020.ACCT, then you can include any column from the ACCT table in the Axiom query. In this case, GL2020.ACCT is known as a *validated column*, ACCT.ACCT is known as a *lookup column*, and the table ACCT is known as a *lookup table*. Lookup relationships are defined when creating or editing table columns.

The reference table can be used in the field definition, to bring in information from the reference table. For example, you could enter `ACCT.Description` and `ACCT.AcctGroup` into the field definition, to bring in a description for each account and its designated account group.

Reference tables can be used in Axiom query settings that impact the data query, such as the data filter and the sum by setting. For example, you could define a data filter of `ACCT.AcctGroup='Benefits'` to restrict the data to only those accounts belonging to the Benefits group. Reference tables can also be used to define data range filters and column filters.

When using a reference table in an Axiom query setting, you must use fully qualified Table.Column syntax. For example, you must enter `ACCT.Description`, not just `Description`. If you only enter `Description`, then Axiom will assume the primary table, which probably does not contain a column named Description.

Axiom supports multiple levels of lookups. For example, the data table GL2020 can look up to the ACCT table (GL2020.ACCT to ACCT.ACCT), and then the ACCT table can look up to another reference table (ACCT.Category to Category.Category), and so on. You can query data from any lookup reference table, regardless of how many levels deep the lookups extend. For more information on including multiple-level lookups in Axiom query settings, see Using multiple levels of column lookups.

Including additional data tables when the primary table is a data table

When the primary table is a data table, then the Axiom query can include data from additional data tables. By default, queries to additional data tables use *outer joins*.

For example, you might query another data table to compare the results of two different plan scenarios or plan years. If the primary data table is BGT2021, you might also want to bring in data from the BGT2020 table for comparison purposes. The data query will be made against both tables—so if you planned for an account this year that you didn't include last year, that record would still be brought into the sheet, and vice versa.

When multiple data tables are included in an Axiom query, the following settings must use a shared column (or a column in a shared lookup table):

- Sum level
- Data filter

This means that any additional data table included in the query must—at minimum—share one column with the primary table. In most cases, the data tables will share multiple columns, such as the shared validated key columns Dept and Acct. See the detailed topics on each setting for more information on valid entries for queries with multiple data tables.

In most cases, when including an additional data table in an Axiom query, you must use fully qualified Table.Column syntax when referencing literal columns in that table. For example, you must specify `BGT2020.M1` in the field definition instead of just `M1`. If you omit the table name, the primary table is assumed. However, if using alias names, then you can enter just the column alias (for example: `CYB1`), and it will point to the appropriate table.

When referencing shared validated columns in the data tables, such as ACCT, you must either specify the column in the shared lookup table (ACCT.ACCT) or use "column-only" syntax (ACCT). This way the setting applies to all data tables in the query. Note that if there are any possible ambiguities (for example, if the data tables link to two different reference tables that contain a column ACCT), then you must use the fully qualified syntax.

Including additional tables when the primary table is a reference table

When the primary table is a reference table, the field definition can include columns from the following additional tables:

- Any reference table that the primary table has a lookup relationship with (a *lookup table*). This means that the primary table looks up to the reference table, not the other way around.
- Any data or reference table that has a direct lookup relationship to the primary table. This means that the table contains a validated column where the lookup points to the primary table. If the table has a multiple-level lookup to the reference table (where the validated column looks up to a reference table that in turn looks up to the primary table), then it cannot be included.

Generally speaking, lookup reference tables can also be used in other Axiom query settings, such as the sum by and the data filter, whereas other tables cannot. See the detailed topics on the applicable settings for more information on valid entries.

When referencing columns from tables other than the primary table, you must use fully qualified Table.Column syntax, or use an alias name that resolves to a column that is valid for inclusion.

Defining the sum level for an Axiom query

Each Axiom query must have a specified "sum level" that determines the level of summation for the rows in the query. To specify the sum level, list one or more columns in the **Sum By** setting in the Sheet Assistant.

For example, if the sum level is `ACCT.ACCT`, then each record of data would represent the total value for each account. If the sum level is `ACCT.ACCT,DEPT.DEPT`, then the same data is retrieved, but in this case each record of data represents a total for each account/department combination, and therefore more rows (or columns, if the query is horizontal) are inserted into the data range to reach the same overall total.

The sum setting can be a key column or a non-key column. If you are summing by a key column, then all key columns in the field definition should also be listed in the sum setting. For example, if your field definition contains entries for the key columns ACCT and DEPT, then the sum setting should be `ACCT.ACCT,DEPT.DEPT`. If the sum is set to just ACCT, then individual DEPT data would not exist due to the summation, and the DEPT field definition in the sheet would not return data as intended.

On the Control Sheet, the sum by setting is labeled **Sum data by these fields**.

What can be used for the sum by

Valid "sum level" entries depend on the primary table and on other tables included in the query.

Data table as primary table

When the primary table is a data table and no other data tables are included in the query, the following database columns can be used to define the sum level:

- Any column in the primary table. You can use fully qualified Table.Column syntax or just the column name. If just the column name is used, and the column is a validated column, the associated lookup table will be assumed (such as `ACCT.ACCT` for an entry of `ACCT`).
- Any column in a lookup table. You must use fully qualified Table.Column syntax. Multiple levels of lookups can be used.

When the primary table is a data table and additional data tables are included in the query, the following database columns can be used to define the sum level:

- Any shared validated column in the data tables, key or non-key. The column reference must be to the shared lookup column, not to the column in the data tables. For example, you can specify `DEPT` (which assumes the DEPT table), or `DEPT.Dept`, but *not* `GL1.Dept`. (Note that if you do enter `GL1.Dept` in this configuration, and GL1 is the primary table, then behind the scenes the entry will be converted to `Dept.Dept` so that the query can be run.)
- Any column in a shared lookup table. You must use fully qualified Table.Column syntax (such as `DEPT.Region` or `ACCT.Category`). Multiple levels of lookups can be used.
- Any shared non-validated column in the data tables, key or non-key. In this case you must specify *only* the column name—do not use fully qualified syntax. For example, enter just `Detail` where both tables contain a non-validated column named Detail. This results in an intentional ambiguity which allows the specified column to apply to all data tables in the query. However, note that if any lookup tables also contain a column with the same name, the ambiguity will not be allowed and the column will be invalid for use in this context.

Reference table as primary table

If the primary table is a reference table, the following database columns can be used to define the sum level:

- Any column in the primary table. You can use fully qualified Table.Column syntax, or just the column name. If just the column name is used, and the column is a validated column, the associated lookup table will be assumed.
- Any column in a lookup reference table. You must use fully qualified Table.Column syntax. Multiple levels of lookups can be used.

If the primary table is a reference table and one or more data tables are included in the query, then the following requirements apply to the allowed sum level entries:

- If the sum level is a column in the primary table, and that column is a validated column, then you must use fully qualified Table.Column syntax.

  For example, if the primary table Dept contains a validated column Region, and you want to use that column as the sum level, then you must enter this as `Dept.Region`. If instead the entry is just `Region`, then the lookup column will be assumed (such as `Region.Region`), which will make the query invalid because the table Region cannot be joined to the data table.

- If the sum level is a column in a lookup reference table, the entry must use multiple-level lookup syntax that starts with the primary table.

  For example, if the primary table Dept contains a validated column Region, and you want to use `Region.RegionType` in the lookup table as the sum level, then you must enter this as `Dept.Region.RegionType`. If instead the entry is just `Region.RegionType`, this will make the query invalid because the table Region cannot be joined to the data table.

You can specify multiple columns, separated by commas. The order of the columns determines the order of the Axiom query keys in the *data control column* for data updates.

Order of sum by columns

If you are using multiple sum by columns, the order of the columns is not important in terms of the returned data. Whether the sum by is `ACCT.ACCT,DEPT.DEPT` or `DEPT.DEPT,ACCT.ACCT`, the same data is returned.

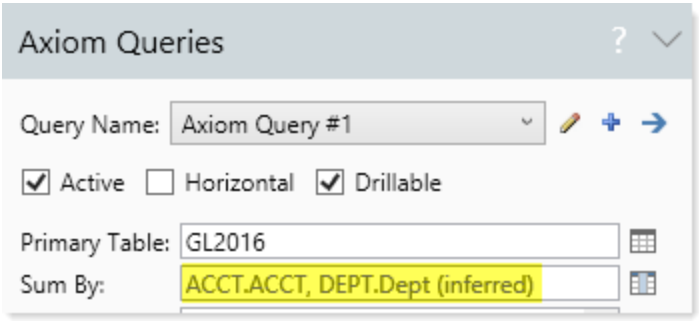
The only thing the column order determines is the order of the codes placed in the data control column. For example, imagine that one record of data is summed by Dept 40000 and Acct 2000. If the sum by is `ACCT.ACCT,DEPT.DEPT`, the codes are written as 2000;40000. If the sum by is `DEPT.DEPT,ACCT.ACCT`, the codes are written as 40000;2000.

This order is only important if the Axiom query is using update behavior, and you are manually inserting rows into the range (such as by inserting a calc method into a plan file). In this example, the calc method must be designed to place the correct codes into the data control column, in the correct order. If the codes are in a different order than the sum by order, the row will not be updated because the Axiom query will not find matches for the codes. For more information, see Manually placing data control codes for update queries.

Omitting the sum by column

If no sum level is specified for a query, then by default the data is summed using the key columns in the primary table. If the primary table is the only data table in the query then all keys are used; whereas if multiple data tables are in the query then only validated keys are used. For example, if the primary table GL2020 has two validated key columns, DEPT and ACCT, then the default sum level is `DEPT.DEPT,ACCT.ACCT`.

When the sum level is not explicitly defined, the assumed sum level displays in the Sheet Assistant in gray font and with the text **(inferred)**.



Defining a data filter for an Axiom query

The **Data Filter** setting for an Axiom query limits the data to be queried from the database. Technically speaking, the data filter is the WHERE clause of the data query.

For example, you might only want to query data for a particular department, region, or manager. Or you might want to exclude specific departments, accounts, or account categories from the query.

The data filter uses standard filter criteria syntax.

NOTE: A user's table filters (as defined in Security) are always applied to an Axiom query, in addition to any data filter. Therefore, the data returned by an Axiom query may vary, depending on the user's security settings.

What can be used in the data filter

Valid data filter entries depend on the primary table and on other tables included in the query.

When the primary table is a data table, and no other data tables are included in the query, the following database columns can be used to define the data filter:

- Any column in the primary table. You can use fully qualified Table.Column syntax or just the column name. If just the column name is used, and the column is a validated column, the associated lookup table will be assumed (such as `ACCT.ACCT` for an entry of `ACCT`).
- Any column in a lookup table. You must use fully qualified Table.Column syntax. Multiple levels of lookups can be used.

When the primary table is a data table and additional data tables are included in the query, the following database columns can be used to define the data filter:

- Any shared validated column in the data tables, key or non-key. The column reference must be to the shared lookup column, not to the column in the data tables. For example, you can specify `DEPT` (which assumes the DEPT table), or `DEPT.Dept`, but *not* `GL1.Dept`.

- Any column in a shared lookup table. You must use fully qualified Table.Column syntax (such as `DEPT.Region` or `ACCT.Category`). Multiple levels of lookups can be used.
- Any shared non-validated column in the data tables, key or non-key. In this case you must specify *only* the column name—do not use fully qualified syntax. For example, enter just `Detail` where both tables contain a non-validated column named Detail. This results in an intentional ambiguity which allows the specified column to apply to all data tables in the query. However, note that if any lookup tables also contain a column with the same name, the ambiguity will not be allowed and the column will be invalid for use in this context.

If the primary table for the query is a reference table rather than a data table, then only columns in the primary table and any lookup reference tables can be used. Data tables cannot be used in the data filter, even if the data table is eligible to be included in the field definition.

Most filters use a reference table to filter the data. For example:

`DEPT=2000` limits the data to department 2000.

`DEPT.Region='North'` limits the data to departments belonging to the North region.

Data filters for templates

When you create plan files from templates, you want to query the data for each individual plan code. Therefore the data filter for the Axiom query must contain a statement that limits the data to the current plan code.

The best way to do this is to reference the plan code somewhere in the file by using `=GetFileGroupProperty("PlanFile")`. Then, reference that cell in your data filter. For example:

```
="DEPT="&Budget!C3
```

If the plan code table is DEPT and the GetFileGroupProperty function is located in cell C3 of the Budget sheet in a template, this example filter dynamically limits data according to the current plan code. For example, if the current plan file is for department 100, this filter would resolve to `Dept=100`.

Remember, you can have multiple sheets and multiple Axiom queries in a template. Only those queries that relate to the plan data for the current plan code must have a data filter to restrict the data by plan code. You can have additional queries in the template that query data from any table, using any filter (or no filter).

Other filtering options for Axiom queries

The Axiom query data filter affects the data returned for the entire query. If you want to filter only specific areas of the query, you can do one or both of the following:

- **Column filters:** You can place a filter on a database column code in the field definition, to filter the data queried from that particular column. For more information, see Filtering the data coming into a spreadsheet column.

- **Data range filters:** You can place a filter on a data range in the insert control column or row, to filter the data displayed in that data range. For more information, see Filtering the data in a data range.

Sheet filters are always honored when running Axiom queries. If you want to filter all of the data queries on a sheet, including GetData functions, you can define a sheet filter. For more information, see Defining sheet filters.

Defining the sort for an Axiom query

You can sort the data for an Axiom query in one of two ways:

- A database sort, which sorts the data before it is inserted into the data range (**Data sort**)
- A spreadsheet column sort, which sorts the data after it is inserted into the data range (**Sheet sort**)

The database sort only applies when data is first inserted into the data range. It does not apply when updating existing data. If data already exists in a data range, and the refresh behavior is update and insert instead of rebuild, then any new records are inserted at the end of the range, and a sort is not applied.

The spreadsheet column sort applies every time a refresh is performed.

If neither sort setting is defined, the data is sorted by the columns specified in the **Sum By** setting, in ascending order.

Database sort

To set a database sort, enter one or more database column names into the **Data Sort** setting in the Sheet Assistant. You can type column names or use the column picker to select and order columns. In the Control Sheet, the corresponding setting is labeled **Sort by database columns**.

In all cases, the database sort level can use the same column(s) defined as the sum level for the query. If a column is valid to define the sum level, it is also valid to define the sort level, as long as both entries are the same. For more information on valid columns for the sum level, see Defining the sum level for an Axiom query. However, the sort level is not required to be the same as the sum level.

Valid database sort entries depend on the primary table and on other tables included in the query.

When the primary table is a data table and no other data tables are included in the query, the following database columns can be used to define the sort level:

- Any column in the primary table. You can use fully qualified Table.Column syntax or just the column name. If just the column name is used, and the column is a validated column, the associated lookup table will be assumed (such as `ACCT.ACCT` for an entry of `ACCT`).
- Any column in a lookup table. You must use fully qualified Table.Column syntax. Multiple levels of lookups can be used.

When the primary table is a data table and additional data tables are included in the query, the following database columns can be used to define the sort level:

- Any shared validated column in the data tables, key or non-key. You can use column-only syntax (such as `DEPT`, which assumes the DEPT table in this context), or fully qualified Table.Column syntax (such as `DEPT.Dept`).
- Any column in a shared lookup table. You must use fully qualified Table.Column syntax (such as `DEPT.Region` or `ACCT.Category`). Multiple levels of lookups can be used.
- Any column in any of the data tables included in the query. You must use fully qualified Table.Column syntax (such as `GL2020.TOT` to sort by total values in that table). If you enter just a column name, then the entry will be interpreted as a column on the primary table.
- Any shared non-validated column in the data tables, key or non-key—but *only if* the same column is also used as the sum level. In this case you must specify *only* the column name—do not use fully qualified syntax (such as `Detail` where all data tables contain a non-validated column named Detail). (Note that if the shared non-validated column is not used as the sum level, and you enter just the column name, then the entry will be interpreted as the column on the primary table.)

If the primary table is a reference table, the following database columns can be used to define the sort level:

- Any column in the primary table. You can use fully qualified Table.Column syntax, or just the column name. If just the column name is used, and the column is a validated column, the associated lookup table will be assumed.
- Any column in a lookup reference table. You must use fully qualified Table.Column syntax. Multiple levels of lookups can be used.

If the primary table is a reference table and one or more data tables are included in the query, then the following requirements apply to the allowed sort level entries:

- If the sort level is a column in the primary table, and that column is a validated column, then you must use fully qualified Table.Column syntax.

  For example, if the primary table Dept contains a validated column Region, and you want to use that column as the sort level, then you must enter this as `Dept.Region`. If instead the entry is just `Region`, then the lookup column will be assumed (such as `Region.Region`), which will make the query invalid because the table Region cannot be joined to the data table.

- If the sort level is a column in a lookup reference table, the entry must use multiple-level lookup syntax that starts with the primary table.

  For example, if the primary table Dept contains a validated column Region, and you want to use `Region.RegionType` in the lookup table as the sort level, then you must enter this as `Dept.Region.RegionType`. If instead the entry is just `Region.RegionType`, this will make the query invalid because the table Region cannot be joined to the data table.

You can specify multiple columns, separated by commas. For example, `ACCT.ACCT,DEPT.DEPT` sorts first by account, then by department, both in ascending order (the default).

If you have defined a unique name for a column that uses special syntax (such as AxAggregate or column filters), then you can use that unique name to define the sort level as long as the column would otherwise be valid for use as detailed previously.

Spreadsheet column sort

To set a spreadsheet sort, enter one or more spreadsheet column letters into the **Sheet Sort** setting in the Sheet Assistant. In the Control Sheet, the corresponding setting is labeled **Sort results by these columns**.

You can sort by any column in the sheet. You can specify multiple columns, separated by commas. For example, `C,D` sorts first by column C, then by column D, both in ascending order (the default).

NOTE: The spreadsheet column sort is only supported for use with standard vertical Axiom queries. This setting is ignored for horizontal queries.

Keep in mind that the column sort is applied every time the data is refreshed, which may impact performance unnecessarily. Also, certain queries are not suited to a column sort—for example, when inserting rows in a plan file that use a multi-row calc method.

Specifying ascending or descending order

You can specify ascending or descending sort order by placing `Asc` or `Desc` after the database column name or the spreadsheet column letter. If omitted, the sort is ascending order by default. For example:

```
Acct Asc; NYB1 Desc
```

This example sorts first by accounts in ascending order, then by Next Year Budget values in descending order.

Using multiple levels of column lookups

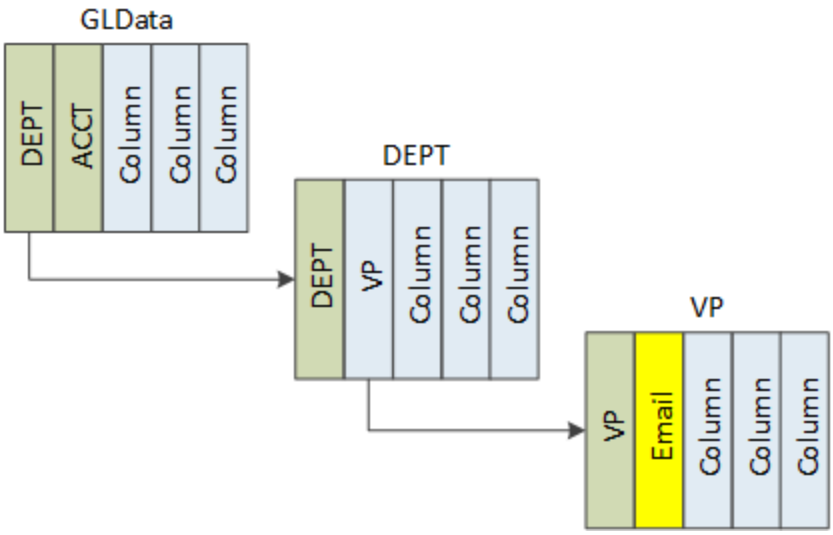
Axiom supports multiple levels of column lookups in tables. These multi-level relationships can be used when querying data in Axiom—from returning data via an Axiom query, to displaying a list of items using GetDataElement or a RefreshVariables data source.

For example, the column `GLData.DEPT` can link to `DEPT.DEPT`, and therefore you can include any column in the DEPT table when querying the GLData table, such as `DEPT.Region` or `DEPT.VP`. This is the first level of column lookup that is used by every system. But the DEPT table itself can have additional lookups, such as `DEPT.VP` linking to `VP.VP`. In this case, you can also include any column in the VP table when querying the GLData table, such as `VP.FullName` or `VP.Email`.

When including a column from a reference table that the data table directly links to (single level of lookup), then you can use normal Table.Column syntax, such as `DEPT.VP`. However, when including a column from a reference table that is linked by multiple levels of lookups, you must indicate the full "path" from the first level of lookup all the way to the target column. For example, to include VP.Email in this Axiom query, you would use the following:

```
Dept.VP.Email
```

The following diagram illustrates the linking between these tables, and the ultimate column to be returned (in yellow):



The easiest way to ensure you are using the correct syntax in this situation is to use the Axiom wizards and tools to select the column. For example, use the column choosers in the Sheet Assistant to complete Axiom query settings. To complete settings in the sheet itself, such as field definition entries, drag and drop the column from the Columns / Aliases section of the Sheet Assistant to the desired cell in the sheet.

If you want to manually type in the column, keep in mind that the syntax can get fairly complicated if there are multiple levels of lookups, or if there are special issues such as:

- Multiple columns in a table have a lookup relationship with the same reference table. For example, `Encounter.PrimaryPhysician` and `Encounter.SecondaryPhysician` both look up to `Physician.Physician`.

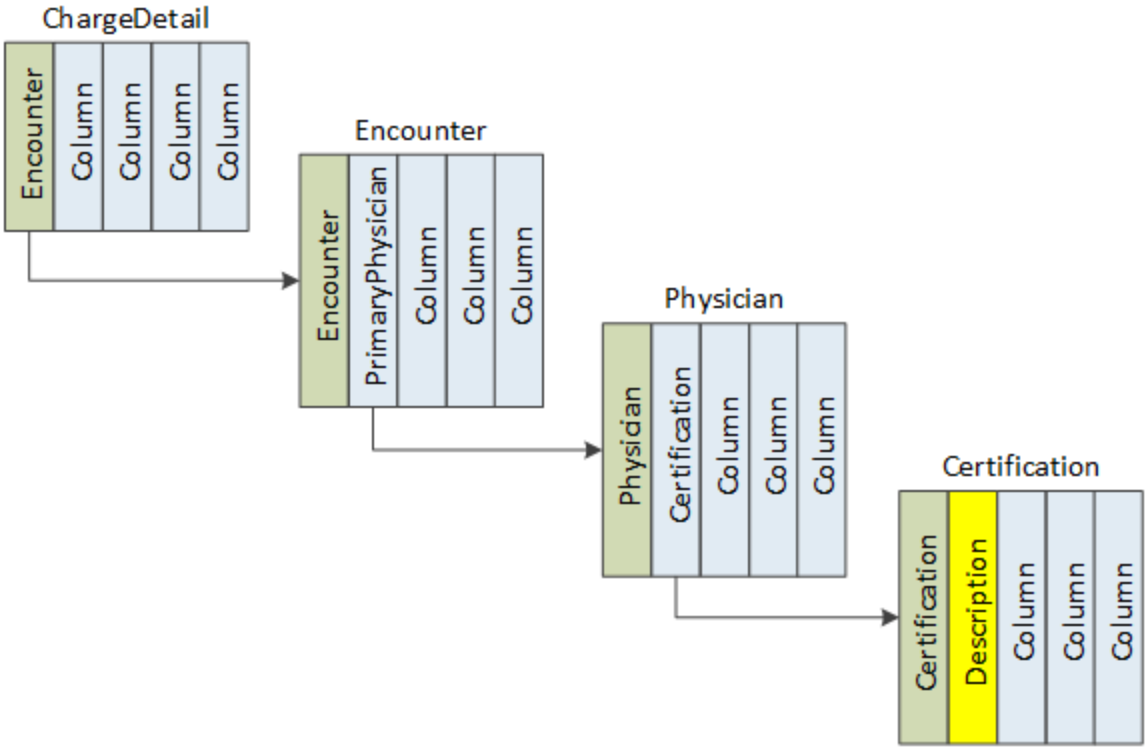
  In this case, you must start the syntax with the name of the column in the original table, instead of starting it with the lookup reference table. You can't just say `Physician.FirstName`, because there are two possible paths to the Physician lookup column. Instead you would have to specify one or the other, such as `PrimaryPhysician.FirstName` (or `Encounter.PrimaryPhysician.FirstName`, if Encounter is not the primary table.

- The validated column does not use the same name as the lookup column. In this case, if there are multiple levels of lookups then you must use the name of the validated column in place of the lookup column. (If there is only one level of lookup, then you can use direct Table.Column syntax for the column in the lookup table.)

The following is an example of several levels of lookups:

```
Encounter.PrimaryPhysician.Certification.Description
```

The following diagram illustrates the linking between these tables, and the ultimate column to be returned (in yellow). In this example, the data table being queried is the ChargeDetail table.



Note the following about this example:

- The syntax starts with the first reference table: Encounter. Because the validated Encounter column in the data table (ChargeDetail) has the same name as its lookup column in the Encounter table, we do not need to include that column.
- The syntax specifies PrimaryPhysician instead of Physician, because the column name in the Encounter table is different than the corresponding column in the Physician table.

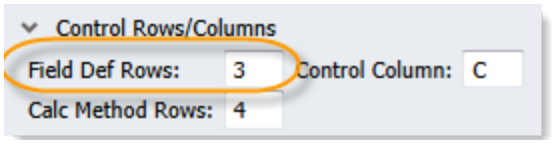
Creating the field definition for an Axiom query

The field definition determines which database columns will be queried and where that data will be placed on the sheet. The field definition can be multiple rows or columns, to accommodate multiple-row and multiple-column calc methods.

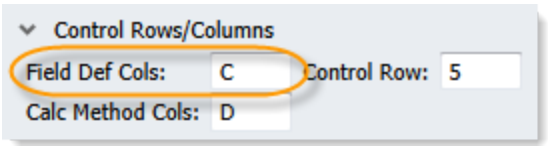
Specifying the location of the field definition

The location of the field definition is defined on the file's Control Sheet. If the query is a standard vertical query, you specify one or more field definition rows. If the query is a horizontal query, you specify one or more field definition columns.

You can use the Sheet Assistant to define these settings, or you can manually edit the Control Sheet. The Sheet Assistant updates dynamically depending on whether the query is vertical or horizontal:

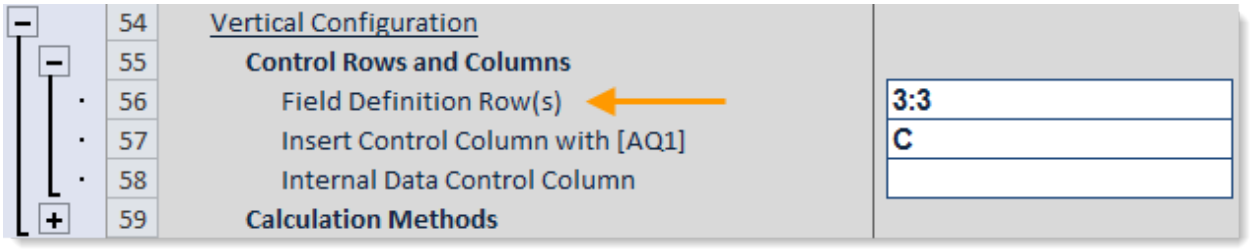


Vertical query

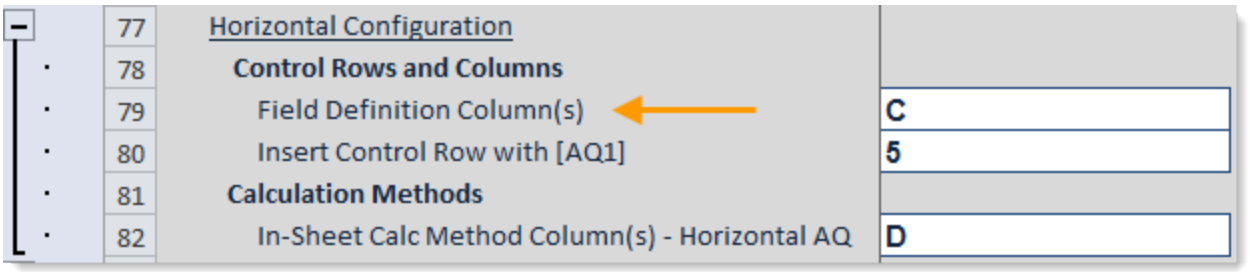


Horizontal query

In the Control Sheet, you must expand the appropriate section to reach the setting, either **Vertical Configuration** or **Horizontal Configuration**.



Vertical configuration



Horizontal configuration

Keep in mind that once you have specified a row or a column location, it does not update automatically if you insert rows or columns into the sheet, or if you move your field definition codes in the sheet. You may want to first design the query on the sheet, figuring out where you want to place items such as headers and subtotals (if needed), and then complete the query settings. If you do change the location in the sheet, you must update the Control Sheet setting.

Creating the field definition on the sheet

To bring data into the target sheet, you must populate the field definition row or column with database column names. When the file is refreshed, data for the specified database column will be inserted or updated into the sheet, within the boundaries of the data range tags (as defined in the insert control column or row).

The following entries can be made in the field definition:

Valid Entry	Description
Column names from the primary table	You can query data from the primary table for the query. You can use fully qualified Table.Column syntax, or "column-only" syntax. If the table name is omitted then the primary table is assumed, unless the column is a validated key column, in which case the lookup table is assumed. For example, `DEPT` is interpreted as `DEPT.DEPT`, but `M1` is interpreted as `GL2020.M1`. This enables querying data from multiple data tables. You can also use column-only syntax for shared non-validated key columns such as Detail, if that column is also used as the sum level for the query. For example: `ACCT`, `DEPT`, `M1` (or `GL2020.M1`), `YTD`
Column names from a lookup reference table	You can query data from any reference table that the primary table has a lookup relationship with. If multiple data tables are used in the query, then all of the data tables must have lookup relationships with the reference table. If the column is a lookup column for a validated key column in the primary table, then "column-only" syntax can be used. For all other columns in the reference table, fully qualified Table.Column syntax must be used. If the query pulls data from multiple data tables, then all of the data tables must link to the reference table. For example: `ACCT`, `ACCT.Description` **NOTE:** Reference tables can have multiple levels of lookups, all of which are eligible to be included in the Axiom query. For more information, see Using multiple levels of column lookups.

Valid Entry	Description
Column names from a table other than the primary table	If the primary table is a data table, then you can query data from any other data table that is eligible to be joined to the primary table (based on the sum by level of the query). Fully qualified Table.Column syntax must be used. **NOTE:** If you are querying data from more than one data table, you should use Table.Column syntax for all data columns in the field definition (or alias names), so that it is clear at a glance which table is being queried for each column. For example: `GL2019.M1` If the primary table is a reference table, then you can query data from any other data or reference tables that directly look up to the primary table. Multiple-level lookups are not supported.
Column alias names	You can use column alias names instead of literal column names. Enter only the alias name, and it will automatically point to the assigned table and column for the alias. If the alias name is later edited to point to a different table column, the query will pull from the new column the next time the query is refreshed. Note that if an alias has the same name as a literal column in the primary table, the literal column is used instead of the alias. We recommend keeping alias names unique from column names to avoid any ambiguity. For example: `NYB1`, `CYA1`, `CYAYTD`

The entries in the field definition must logically complement the structure of the calc methods used in the Axiom query (either an in-sheet calc method or the sheet's calc method library). For more information, see How calc methods work with field definitions.

The field definition must contain only valid database column names (including aliases), and reserved Axiom tags for other worksheet processes. Any other content in the field definition will cause an error when the Axiom query is run.

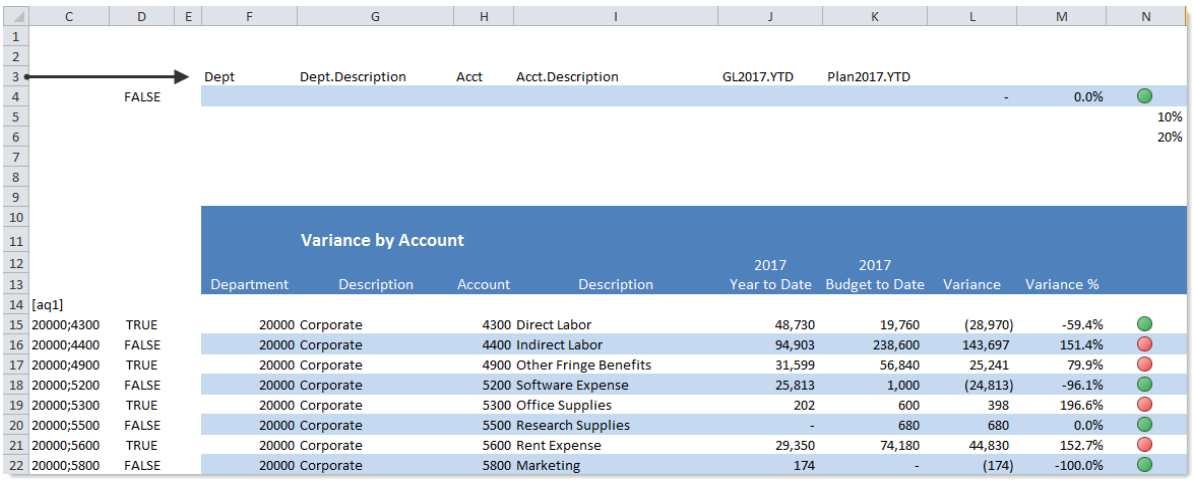
Any reserved Axiom tags placed in the field definition will be ignored by the Axiom query. For example, you can place the tag `[ExportToFile]` in the field definition row, so that you can use the same row for two purposes (as the field definition row for the Axiom query, and as the control row for an export-to-file process).

NOTES:

- Field definition entries cannot be placed within the data control column or row (which is usually the same as the insert control column or row).
- If the primary table is not explicitly specified on the Control Sheet, then Axiom will assume the primary table based on the first data column in the field definition. (Reference table columns are ignored, unless all of the columns are from a reference table.) Generally, this is intended for situations where you are using alias names in the field definition, so that the primary table can dynamically change depending on the table that the alias names point to. The data column must be an alias name or a fully qualified Table.Column name to assume the primary table.

Field definition example

In this example, row 3 is the field definition row for a standard vertical Axiom query:



	C	D	E	F	G	H	I	J	K	L	M	N
1												
2												
3				Dept	Dept.Description	Acct	Acct.Description	GL2017.YTD	Plan2017.YTD			
4		FALSE								-	0.0%	
5												10%
6												20%
7												
8												
9												
10				Variance by Account								
11												
12								2017	2017			
13				Department	Description	Account	Description	Year to Date	Budget to Date	Variance	Variance %	
14	[aq1]											
15	20000;4300	TRUE		20000	Corporate	4300	Direct Labor	48,730	19,760	(28,970)	-59.4%	
16	20000;4400	FALSE		20000	Corporate	4400	Indirect Labor	94,903	238,600	143,697	151.4%	
17	20000;4900	TRUE		20000	Corporate	4900	Other Fringe Benefits	31,599	56,840	25,241	79.9%	
18	20000;5200	FALSE		20000	Corporate	5200	Software Expense	25,813	1,000	(24,813)	-96.1%	
19	20000;5300	TRUE		20000	Corporate	5300	Office Supplies	202	600	398	196.6%	
20	20000;5500	FALSE		20000	Corporate	5500	Research Supplies	-	680	680	0.0%	
21	20000;5600	TRUE		20000	Corporate	5600	Rent Expense	29,350	74,180	44,830	152.7%	
22	20000;5800	FALSE		20000	Corporate	5800	Marketing	174	-	(174)	-100.0%	

- Wherever a database column code is placed in the field definition row, data from that column is brought into the data range (the section beginning with `[aq1]`).
- Two data tables are included in this Axiom query, GL2017 and Plan2017. Instead of the fully qualified column names in the above example, alias names could have been used, such as CYA_YTD and CYB_YTD.
- Column-only syntax is used to bring in DEPT and ACCT. Axiom intelligently interprets these entries as DEPT.DEPT and ACCT.ACCT, so that they apply to both data tables. To bring in the department and account descriptions, fully qualified syntax must be used (DEPT.Description).

Filtering the data coming into a spreadsheet column

You can filter the data coming into a particular spreadsheet column by appending a filter criteria statement to the database column code in the Axiom query field definition. This is known as using a *column filter* or a *field definition filter*. For example:

```
CYA1;DEPT.Region='North'
```

Use a semi-colon to separate the filter criteria statement from the database column code.

When the file is refreshed, the data coming into this column of the spreadsheet (or row, if it is a horizontal query) would be limited to departments assigned to the North region. You might have several columns like this, each one referencing the same database column but restricting the data to a specific region, or manager, etc., for comparison purposes.

D3 | cya1;dept.region='us west'

	C	D	E	F	G
2					
3	acct.category	cya1;dept.region='u	cya1;dept.regio	cya1;dept.region='us central'	
4					$ -
5	**January 2011**				
6					
7	**Regional Comparison**	**US West**	**US East**	**US Central**	**Total**
8					
9	Marketing	$ 4,861.66	$ 16,518.19	$ 17,171.26	$ 38,551.11
10	Other Expenses	$ 50,936.59	$ 59,631.96	$ 56,329.31	$ 166,897.86
11	Other Income/Expense	$ -	$ 33,616.51	$ -	$ 33,616.51
12	Overhead	$ 113,261.92	$ 217,558.19	$ 102,989.65	$ 433,809.76
13	Payroll	$ 802,950.01	$ 913,342.13	$ 737,266.98	$ 2,453,559.12
14	Travel	$ 411,268.29	$ 230,408.65	$ 275,793.70	$ 917,470.64

Example report using column filters

Column filters can be based on a lookup reference table, or on the table that the field definition entry belongs to. You can use fully qualified Table.Column syntax, or "column-only" syntax. If the table name is omitted then the primary table is assumed, unless the column is a validated key column, in which case the lookup table is assumed.

Column filters can use compound filters and parentheses in the filter criteria statement.

NOTE: When the primary table is a reference table, column filters can be used on data tables included in the field definition. In the filter, you can use any lookup reference table for the data table, even though that reference table has no relationship to the primary table. For example, if the primary table is Dept, you can include a column filter such as `GL2020.M1;Acct.Category='Revenue'` to filter the data table column, even though Dept has no lookup relationship with the Acct table.

Naming a filtered column

You can optionally name a filtered column, so that it can be used in the following Axiom query features:

- Data Sort
- Post-Query Filter

Defining a name allows you to use the specific filtered results of the column in these settings, instead of the unfiltered results. For example, you might repeat the same column several times in the field definition, but each time apply a different filter. If you use just the column name in an Axiom query setting, then the setting uses the unfiltered results of the column. But if you define a unique name for a filtered column and then use the unique name in the setting, the setting then uses the filtered results for that specific named column.

To name a filtered column, append a unique name to the base column name using a colon. For example:

```
GL2020.M1:Jan19;Acct.Category='Revenue'
```

In this example, we have applied a filter to the `GL2020.M1` column, and then named this filtered column `Jan19`. The pseudo-column name Jan19 can then be used in the Axiom query settings listed above.

Specifying an alternate aggregation method for a field definition

By default, when you query data using an Axiom query, the results are aggregated as follows:

- Numeric columns are summed based on the **Sum by** level for the query. "Numeric" in this context means any column with a data type of Integer (all types) or Numeric. The following exceptions apply:
  - Key columns, validated columns, and columns with a column classification of Dimension are not summed; instead the maximum value is returned.
  - If the numeric column is on a lookup table, and the column classification is Value, then special summing behavior is applied (LookupSum). The values are summed based on the rows returned from the lookup table versus the rows returned from the primary table. For more information, see Lookup aggregation types.
- All other column types return the maximum value. In most cases, the **Sum by** for the query is set so that there is only one possible value per record.

In certain cases you may want to return different information, such as a count of records. This option is specified at the database column level, in the field definition. The syntax for specifying an alternate aggregation is as follows:

```
AxAggregate(AggregationType)ColumnName
```

The entire field definition entry cannot have any spaces. The column name can use fully-qualified syntax, column-only syntax, or alias names.

NOTE: Alternate aggregation is not supported when the primary table for the query is a system table, such as Axiom.Aliases.

Standard aggregation types

The following aggregation types are supported for use with AxAggregate:

- `RowCount`: Returns the count of records in the query. The codes `Count` and `RCount` are also valid and return the same result.
- `DistinctCount`: Returns the count of unique records in the query. For example, if the query returns two records that both have a value of 10, RowCount will return 2 but DistinctCount will return 1.
- `RowAvg`: Returns the average value of records in the query. Does not apply when querying string or date columns. The codes `Avg` and `RAvg` are also valid and return the same result.
- `RowSum`: Returns the sum of records in the query. Does not apply when querying string or date columns. The codes `Sum` and `RSum` are also valid and return the same result.
- `Min`: Returns the minimum record in the query.
- `Max`: Returns the maximum record in the query.

For example, if you want to return a count of unique records queried from the CPREQ2020 table, you would enter the following syntax into the field definition row or column:

```
AxAggregate(DistinctCount)CPREQ2020.CAPREQ
```

When using the standard aggregation types, the results are based on the primary table if the specified column is from the primary table or from a lookup table. If the column is on a lookup table, you should consider whether to use a lookup aggregation type instead, depending on the desired results. If the specified column is on another data table, then the results are based on that table.

Lookup aggregation types

Special lookup aggregation types are available for use when the column you want to aggregate is on a lookup reference table. This means you have a primary table with a lookup to a reference table, and you want to aggregate a column on that lookup reference table.

When using these special aggregation types, the results are based on the lookup table instead of on the primary table.

- `LookupCount`: Returns the count of records returned from the lookup table. The code `LCount` is also valid and returns the same result.
- `LookupAvg`: Returns the average value of records returned from the lookup table. The code `LAvg` is also valid and returns the same result.
- `LookupSum`: Returns the sum value of records returned from the lookup table. The code `LSum` is also valid and returns the same result.

Imagine that you want to return the average of a column on a reference table that holds a numeric value, such as `CPREQ2020.UnitCost`, and that reference table is not the primary table. In this case, using the standard aggregation type of RowAvg would return the average based on the query results from the primary table, which likely is not the expected average value. Instead you can use LookupAvg to return the average based on the query results from the lookup table.

To illustrate the difference, imagine that the CPREQ2020 reference table has the following values:

CPREQ2020CAPREQ	CPREQ2020UnitCost
27	100.00
32	25.00

The average UnitCost of this data is (100+25)/2, which is 62.5. In most cases this would be the expected average for the unit cost value

However, when the primary table of the query is a different table—such as CPData2020—then standard aggregation types are calculated on data that appends the lookup reference table values to each row of the primary table. This is an issue because in the primary table, the CAPREQ keys may be listed multiple times, causing the appended lookup values such as unit cost to be repeated. For example, the values returned by the query could look like this, where CAPREQ 27 is listed multiple times:

CPData2020.ACCT	CPData2020.CAPREQ	CPREQ2020.UnitCost
1000	27	100.00
1000	32	25.00
2000	27	100.00

In this case, what RowAvg returns is (100+25+100)/3, which is 75. In order to get the expected average based on the lookup reference table instead of the primary table, you must use LookupAvg—which returns 62.5.

This issue does not apply when the reference table is the primary table of the query. If the primary table was CPREQ2020, then RowAvg would be calculated on the primary table and return the expected value of 62.5.

Using column filters with AxAggregate

You can use column filters and AxAggregate together on a field definition entry, so that the alternate aggregation is applied to a subset of data (as defined by the filter), instead of all the data returned by that column. For example:

```
AxAggregate(Min)GL2020.m1;dept.region='US West'
```

This example returns the minimum values for GL2020.m1, for departments that belong to the US West region only, at the "sum by" level of the query. This might be used in a report that has several columns like this, each one using a different region filter for purposes of comparing the region values side by side.

All aggregation types can be used in combination with column filters.

Naming a column that uses alternate aggregation

You can optionally name a column that uses AxAggregate, so that it can be used in the following Axiom query features:

- Data Sort
- Post-Query Filter

Defining a name allows you to use the alternate aggregated results of the column in these settings, instead of the default aggregated results. If you use just the column name in an Axiom query setting, then the setting uses the default aggregation. But if you define a unique name for the AxAggregate column and then use the unique name, the setting then uses the alternate aggregation for that specific named column.

To name a column that uses AxAggregate, append a unique name to the base column name using a colon. For example:

```
AxAggregate(DistinctCount)CPREQ2020.CAPREQ:ReqCount
```

In this example, we have applied alternate aggregation to the `CPREQ2020.CAPREQ` column, and then named this filtered column `ReqCount`. The pseudo-column name ReqCount can then be used in the Axiom query settings listed above.

Creating the insert control column or row for an Axiom query

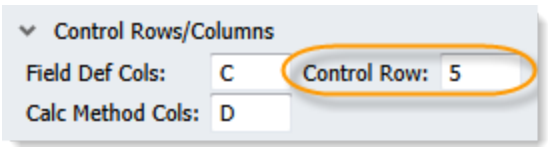
The insert control column or row defines the ranges where data will be inserted (or updated) on the target sheet. Each Axiom query can have one or more data ranges within an insert control column or row.

Specifying the location of the insert control column or row

The location of the insert control column or row is defined on the file's Control Sheet. If the query is a standard vertical query, you specify an insert control column. If the query is a horizontal query, you specify an insert control row.

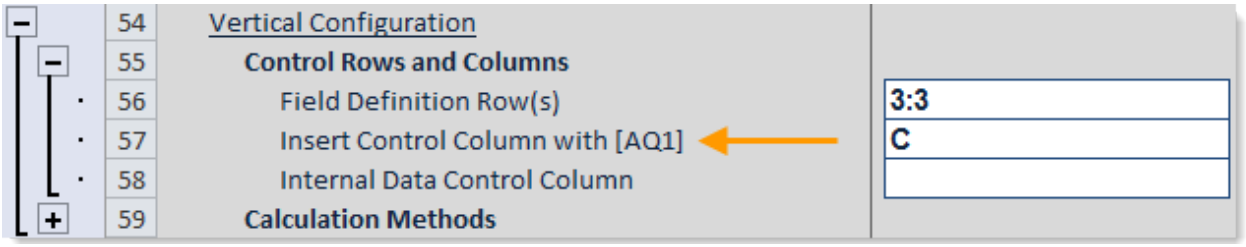
You can use the Sheet Assistant to define these settings, or you can manually edit the Control Sheet. The Sheet Assistant updates dynamically depending on whether the query is vertical or horizontal:

Vertical query

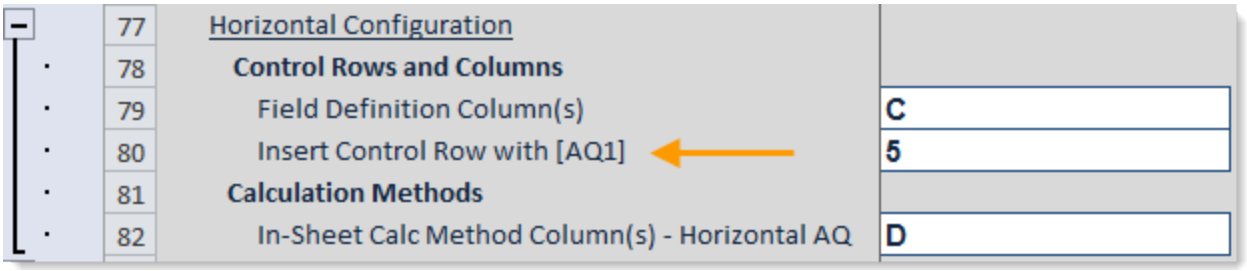


Horizontal query

In the Control Sheet, you must expand the appropriate section to reach the setting, either **Vertical Configuration** or **Horizontal Configuration**.



Vertical configuration

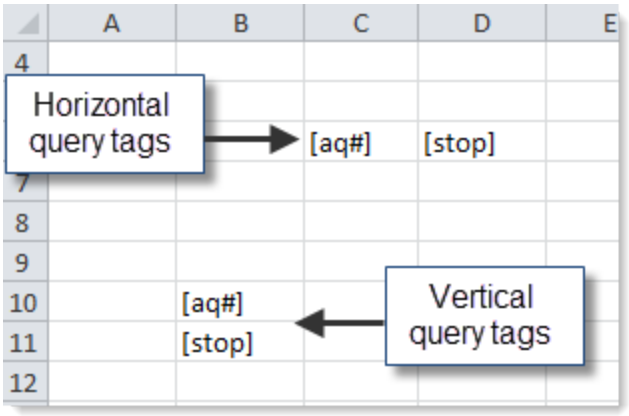


Horizontal configuration

Keep in mind that once you have specified a row or a column location, it does not update automatically if you insert rows or columns into the sheet, or if you move your Axiom query tags in the sheet. You may want to first design the query on the sheet, figuring out where you want to place items such as headers and subtotals (if needed), and then complete the query settings. If you do change the location in the sheet, you must update the Control Sheet setting.

Placing data range tags in the sheet

The insert control column or row must contain at least one set of data range tags, to specify where the data should be placed in the sheet. To define a data range on the sheet, enter the following tags into consecutive cells within the insert control column or the insert control row (depending on the orientation of the query):



Number (#) represents the number of the Axiom query that this set of tags applies to. For example, if you are placing tags for Axiom query #1, you enter the following:

```
[aq1]
[stop]
```

When the Axiom query is processed, data records are inserted after the cell marked with `[aq#]`. If the query is vertical, the records are inserted below the cell. If the query is horizontal, the records are inserted to the right of the cell.

You can type the tags into the spreadsheet directly, or you can use one of the following tools to automatically insert tags:

- In the Sheet Assistant, click the arrow button to the right of the Axiom query name. This inserts a basic set of tags at the current cursor location, in the insert control column or row.

- Right-click the cell, then select **Axiom Wizards > Insert Axiom Query Data Range > *QueryName***. Click **OK** on the resulting dialog to insert a basic set of tags at the current cursor location, in the insert control column or row.

Each Axiom query can have multiple sets of data range tags within a single insert control column or row. In this case you would define a filter to limit the data for each range, otherwise the data in each range would be identical. For more information, see Filtering the data in a data range.

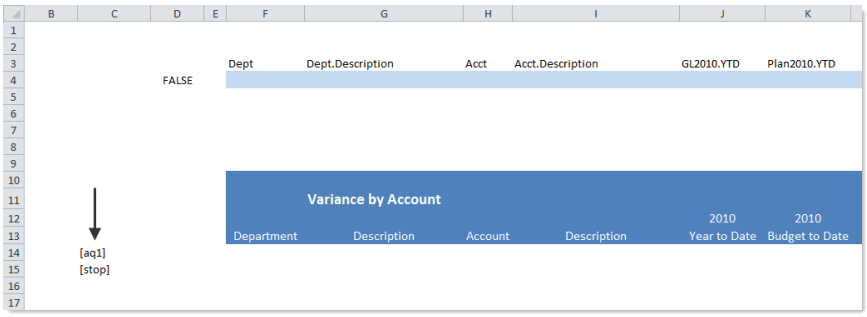
By default, records are populated into data ranges by inserting entire rows or columns. If desired, you can configure the Axiom query's insertion behavior so that it is populated by inserting cells within a designated range instead. If you change the insertion behavior to **InsertRange**, then you must use additional tags in the data range to specify the insertion range: `[Stoprange]` to specify the end of the range, and optionally `[Startrange]` to specify the start of the range. For more information, see Insertion options for Axiom queries and Defining the insertion range when using Insert Range.

NOTES:

- By default, the specified insert control column or row also holds the data control codes for the Axiom query. When data records are inserted by the query, the key codes for each record are placed in the insert control column or row, to be used for subsequent data updates (if applicable). For more information, see About data control codes for Axiom queries.
- The action taken on the data range depends on the refresh behavior for the Axiom query. For example, if the refresh behavior is set to rebuild, then any existing data in the range is always deleted, and the range is repopulated with inserted records. If the refresh behavior is set to insert and update, then existing data is updated, and new rows are only inserted if the record doesn't already exist in the range. For more information, see Specifying how data is refreshed in an Axiom query.
- If you want to dynamically enable or disable an Axiom query, this should always be done by using a formula in the **Active** setting for the query on the Control Sheet. Showing and hiding data range tags using a formula is not a valid way to dynamically enable or disable a query. Active, non-nested Axiom queries should always have at least one set of data range tags.
- Data range tags must always be present in pairs. An `[aq#]` tag must have a `[stop]` tag, and vice versa.

Insert control column example

The following example is for a standard vertical Axiom query. The first screenshot shows the insert control column before the Axiom query has been run. The designated column is column C, and the `[aq1]` tag specifies where the data range will begin.



Once the query is run, the data range is populated with data from the database, as shown in the following screenshot:

	B	C	D	E	F	G	H	I	J	K
1										
2										
3					Dept	Dept.Description	Acct	Acct.Description	GL2010.YTD	Plan2010.YTD
4			FALSE							
5										
6										
7										
8										
9										
10										
11						Variance by Account				
12									2010	2010
13					Department	Description	Account	Description	Year to Date	Budget to Date
14		[aq1]								
15		20000;4300	TRUE		20000	Corporate	4300	Direct Labor	48,730	19,760
16		20000;4400	FALSE		20000	Corporate	4400	Indirect Labor	94,903	238,600
17		20000;4900	TRUE		20000	Corporate	4900	Other Fringe Benefits	31,599	56,840
18		20000;5200	FALSE		20000	Corporate	5200	Software Expense	25,813	1,000
19		20000;5300	TRUE		20000	Corporate	5300	Office Supplies	202	600
20		20000;5500	FALSE		20000	Corporate	5500	Research Supplies	-	680
21		20000;5600	TRUE		20000	Corporate	5600	Rent Expense	29,350	74,180
22		20000;5800	FALSE		20000	Corporate	5800	Marketing	174	-
23		20000;6000	TRUE		20000	Corporate	6000	Other Travel	3,353	15,400
24		20000;6200	FALSE		20000	Corporate	6200	Legal Expense	20,000	-
25		20000;6300	TRUE		20000	Corporate	6300	Professional Services	1,607	-
26		20000;6400	FALSE		20000	Corporate	6400	Membership	2,835	58,020
27		20000;6600	TRUE		20000	Corporate	6600	Filing Fees	259	-
28		20000;6800	FALSE		20000	Corporate	6800	Human Resources	-	3,200

Note that key codes are automatically placed in the insert control column, because this column is also the data control column. If the Axiom query is set to update instead of rebuild, then each subsequent time the query is run, the existing data rows will be updated, based on the key codes in the data control column.

Insert control row example

The following example is for a horizontal query. The first screenshot shows the insert control row before the Axiom query has been run. The designated row is row 3, and the [aq1] tag specifies where the data range will begin.

	A	B	C	D	E	F	G	H
1								
2								
3				→	[aq1]	[stop]		
4		dept						
5								

Once the query is run, the data range is populated with data from the database, as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2													
3				→	[aq1]	10000	11000	20000	21000	22000	23000	24000	25000
4		dept				10000	11000	20000	21000	22000	23000	24000	25000
5													

This is a very simple example intended to contrast the insert control structure of a horizontal query from a vertical query.

Filtering the data in a data range

You can place a filter on an Axiom query tag, to limit the data to be brought into a particular data range. The filter statement must be appended to the end of the Axiom query start tag, separated by a semi-colon. For example:

```
[aq1;ACCT.AcctCategory='Benefits']
```

When the file is refreshed, this range will be limited to data for accounts belonging to the benefits category.

Data range filters can be used to group data into multiple sections, allowing for subtotals and other formatting. For example, you could have multiple data ranges with filters as follows:

```
[aq1;ACCT.AcctCategory='Benefits']
[stop]

[aq1;ACCT.AcctCategory='Compensation']
[stop]
```

When the file is refreshed, there will be two data ranges, one containing the benefit accounts, and one containing the compensation accounts. Both ranges use the settings for Axiom query #1, the data range filter simply limits the data displayed in each range.

When to use data range filters

Generally speaking, data range filters should only be used as a means to organize the data in the sheet into multiple sections. It should *not* be used to filter the data coming back from the database.

The data range filter is applied after the data has already been returned from the database, to determine where that data is placed in the sheet. Theoretically, this means that you could query 5,000 rows from the database but only place 100 of those rows in the sheet due to the data range filters. This is known as *unmatched data*, and it should be avoided for performance reasons. If you really only need 100 rows, then you should apply a filter at the database level instead (such as by using the Axiom query **Data Filter**), so that the database query returns only the 100 rows that you need. The data range filters can then be used to determine where those 100 rows are placed in the sheet.

For more information on unmatched data, see Checking for unmatched data in an Axiom query.

Valid data range filters

The data range filter can be based on any column that would be valid in the Axiom query **Sum By** or in the field definition—in fact, in most cases the filter column is also being used in one or both of these places.

You can use fully qualified Table.Column syntax, or "column-only" syntax. If the table name is omitted then the primary table is assumed, unless the column is a validated key column, in which case the lookup table is assumed.

NOTE: When querying multiple data tables, it is not possible to use a shared, non-validated column in the data range filter and have it apply to all of the data tables. This type of ambiguity is allowed in the main Axiom query **Data Filter**, but it is not supported in data range filters. The non-validated column must be fully qualified in this context.

It is recommended to include the filter column in the sum by, to ensure that data is summed at the same level that you want to present it in the sheet. If the filter column is not part of the sum by, then data may not be returned in the data ranges as you expect if the records being summed do not have the same value for the filter column.

If the filter column is not included in the sum by or the field definition, then Axiom will automatically include the filter column from the first data range in the database query, as if it were part of the field definition. This is to ensure that the query includes the data necessary to apply the data range filter in the sheet. However, if you have multiple data range filters that use different columns, and all of those columns are not represented in the sum by or in the field definition, then you must enable the following setting in the **Data Options** section of the Control Sheet: **Enable full AQ query validation mode**. This will cause all columns used in data range filters to be included in the database query.

Creating a filtered data range

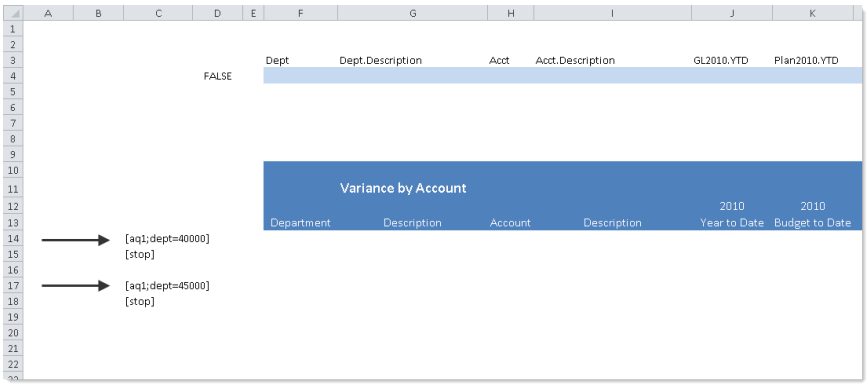
You can manually type the filter into the data range tag, or you can use the data range wizard to automatically create a filtered tag with the correct syntax:

1. Place your cursor in the desired location, and then right-click and select **Axiom Wizards > Insert Axiom Query Data Range > *AxiomQueryName***.
2. In the **Insert Data Range** dialog, type a filter or use the Filter Wizard to build a filter. When you click **OK**, the data range tag will be inserted into the appropriate control column or row, with the filter appended to the tag.

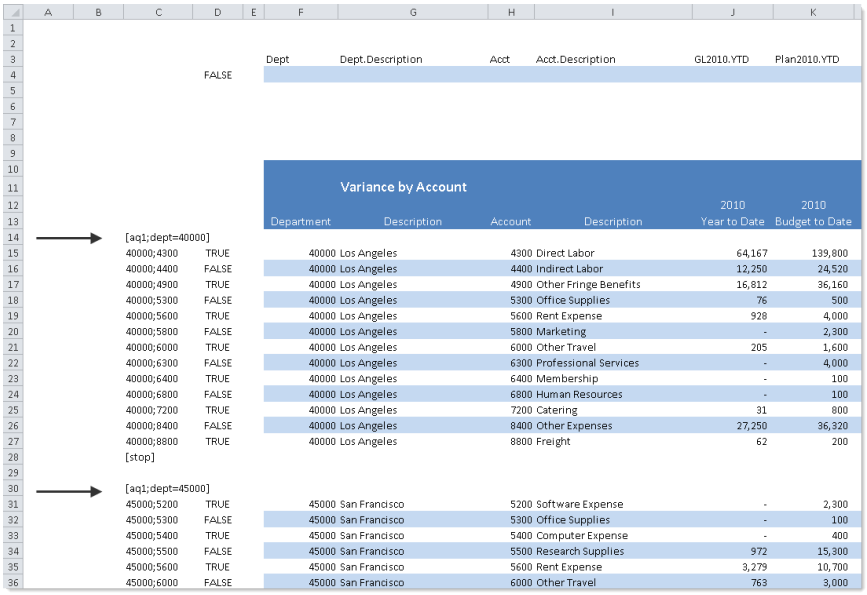
If a tag already exists at a particular location, you can double-click the `[aq#]` tag to add or edit a filter on the existing tag.

Example data range filters

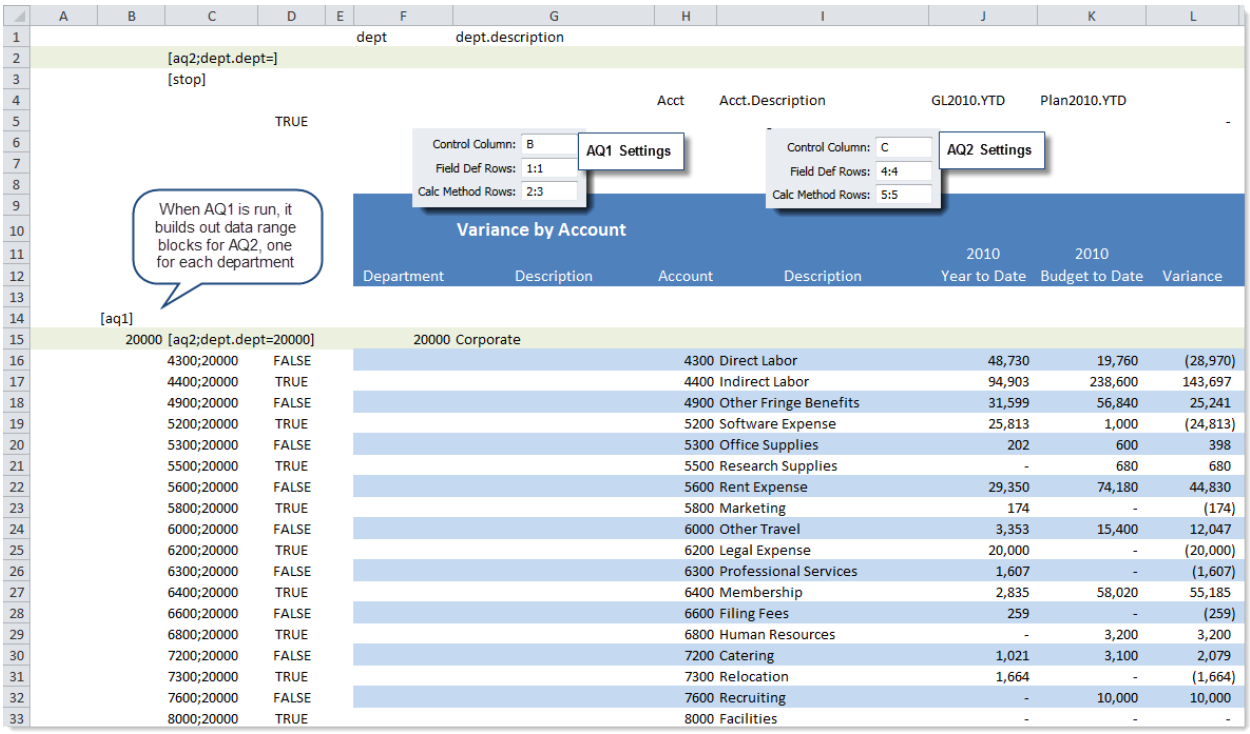
In the following example, there are two data ranges, each with a separate filter:



When the Axiom query is run, each data range is populated only with the data that meets the filter. In this case, the first range displays data for department 40000 and the second range displays data for department 45000.



Although these particular data ranges were "hard-coded," a good way to automate multiple data range filters is to use "nested" Axiom queries. For example, if you wanted this report to contain one section per department, it would be time-consuming and inflexible to manually create each data range filter for each department. Instead, you could have one "parent" Axiom query that brings in departments and dynamically builds out the data ranges for a second "child" Axiom query. The child query would be the query that brings in the account data.



About data control codes for Axiom queries

By default, the insert control column or row for an Axiom query also contains the data control codes for the query. When an Axiom query inserts records into a sheet, the appropriate key codes for each record are placed into this column or row. When an Axiom query updates data in a sheet, these key codes are used to associate each record in the sheet with its corresponding record in the database, so that each row is updated with the appropriate data.

The key codes placed in the sheet correspond to the "sum by" level for the query. For example, if the sum by level is Acct.Acct, then the key codes are account codes. If the sum by level is Acct.Acct;Dept.Dept, then the key codes are the Acct and Dept codes concatenated together.

NOTE: For the purposes of data control for the Axiom query, the "key code" is whatever code corresponds to the level used for the sum by; it does not necessarily mean a code from a literal key column as specified in the table. For example, if the sum by level for the query is ACCT.Category, then the "key codes" are category names as defined in that column, such as "Revenue" or "Benefits."

It is recommended to use the default behavior where the insert control column or row also holds the data control codes. However, the Control Sheet does contain a separate setting to define an **Internal Data Control Column** (for vertical queries only). If this setting is blank, the default behavior applies and the insert control column is used. If the setting contains a column reference, then that column is used for

the data control codes instead. The primary purpose of the separate setting is for backward compatibility when upgrading older Control Sheets—older versions of Axiom required these columns to be defined separately. New queries should use the default behavior unless there is a compelling reason not to.

Regardless of where the data control codes are placed, these codes should be thought of as system-managed and therefore not used as display data for the file or to control other behavior for the file. For example, if the sum by level for the query is Acct.Acct, there should always be a separate column in the field definition that brings in Acct.Acct, and this column should be used to display the account code for each record and to drive any formulas that depend on the account code. Axiom reserves the right to change the structure or display of the data control codes as needed to support Axiom query functionality, which could impact your file if you are relying on these codes for any purpose other than data inserts and updates for the Axiom query.

If the refresh behavior for the query is Update, then you may need to manually place the appropriate data control codes in this column. This is the case if the codes have not been placed by a previous execution of the query using Rebuild or Insert. For more information, see Manually placing data control codes for update queries.

Checking for unmatched data in an Axiom query

In some cases, an Axiom query may return data records from the database that are not placed anywhere on the sheet. This "orphan" data is known as *unmatched data* (or *non-matched data*).

Unmatched data may occur in the following circumstances:

- The data ranges for the Axiom query have filters defined, and the record does not match any of the filters.
- The Axiom query is set to update only, and there are no matching keys for the record in any of the data ranges.

In both cases the data record has nowhere to be placed on the sheet, so nothing is done with the record.

It is a good idea to know if your Axiom query has unmatched data. Too much unmatched data can affect query performance—for example, if the query settings return 5,000 records from the database, but only 200 of those records are brought into the sheet. Assuming that you really only want those 200 records, it is recommended to define a data filter on the query (or as a sheet filter) so that the overall data returned is closer to (or exactly matches) the set of data you want to place in the sheet.

You may also want to check for unmatched data if your intent is to use all records returned by the query, to make sure the sheet is set up correctly. For example, plan files often have several data ranges with filters, to organize the sheet into "blocks" by account type. In this case you may want to check for unmatched data, to ensure that the filters on your data ranges are configured correctly and encompass all of the accounts that you want to bring into the sheet.

To check for unmatched data, you can use the QA Diagnostics tool, or you can use a special data range tag to bring the unmatched data into the sheet. For more information on using QA Diagnostics, see Using file diagnostics for troubleshooting and optimization.

Using the UnmatchedData tag

To use the UnmatchedData tag, create a data range tag as follows:

```
[AQ#;UnmatchedData]
[Stop]
```

Where # is the number of the Axiom query.

The UnmatchedData tag can be used with either vertical queries or horizontal queries (the above example is for a vertical query).

You can use the data range wizard to insert an UnmatchedData tag. To do this, right-click a cell and select **Axiom Wizards > Insert Axiom Query Unmatched Data Range > *AxiomQueryName***. This will place the tag in the appropriate insert control column or row. You can also double-click an existing tag to edit it and then select **Unmatched data range**.

When the query is run, any data records that were returned by the database query but could not be placed in the other data range(s) are placed in the UnmatchedData range.

The UnmatchedData range is a special range intended for testing and troubleshooting only. The UnmatchedData range is always rebuilt and inserts all unmatched data, regardless of the refresh type of the query. For example, if the query is set to update only, all of the regular data ranges will honor the update behavior, but the UnmatchedData range will rebuild and display all records that were not found in the regular data ranges.

Calc methods and Axiom queries

This section discusses requirements and considerations for using calc methods with Axiom queries. Calc methods define the formatting and calculations for records being inserted into the Axiom query's data ranges.

You can use calc methods from the sheet's calc method library, or you can use an in-sheet calc method.

- Calc method libraries are stored per sheet name across a file group, and can only be used in templates/plan files. When using calc method libraries, each record to be inserted can be assigned a different calc method.
- In-sheet calc methods are defined on the target sheet of the Axiom query, in the designated rows or columns as specified on the Control Sheet. In-sheet calc methods can be used in any Axiom file, but are always used in report files. Each Axiom query can have only one in-sheet calc method, so every record inserted uses the same calc method.

Calc methods apply when the Axiom query's rebuild behavior is set to either rebuild or insert. If an Axiom query is update-only, then calc methods are not applied to the data.

How calc methods work with field definitions

Whether you are using an in-sheet calc method or the calc method library for an Axiom query, the calc method and the field definition for the query must logically complement each other.

To help understand this concept, imagine a field definition row layered on top of a calc method row. The combination of both items is how the data records will appear when the file is refreshed. The following is a very simple example:

The *field definition* is set up to bring in data from the database in columns C-F:

C	D	E	F	G
acct.category	lya1	lya2	lya3	

The *calc method* has a sum formula in column G, and then formatting-only in the other columns:

C	D	E	F	G
				$0.00

When the query is refreshed, the calc method is applied to each database record brought in from the query, resulting in the following rows:

C	D	E	F	G
	Jan 2011	Feb 2011	March 2011	Total Q1
Marketing	$ 64,528.95	$ 53,933.90	$ 80,025.54	$198,488.39
Other Expenses	$ 527,685.30	$ 681,040.58	$ 589,552.22	$1,798,278.10
Other Income/Expense	$ 55,831.06	$ 226,622.28	$ 294,850.16	$577,303.50
Overhead	$ 1,501,824.18	$ 1,695,918.74	$ 1,411,840.72	$4,609,583.64
Payroll	$ 4,260,777.28	$ 4,034,241.26	$ 4,224,155.68	$12,519,174.22
Revenue	$ 12,327,243.15	$ 9,587,902.35	$ 8,892,243.20	$30,807,388.70
Taxes	$ 91,727.76	$ 68,451.97	$ 98,059.18	$258,238.91
Travel	$ 4,327,874.10	$ 2,303,203.03	$ 2,114,247.87	$8,745,325.00

NOTE: This example uses a vertical Axiom query, but the same concept applies to horizontal queries.

If there is a conflict between entries in the calc method and the field definition, the field definition takes priority. For example, if a cell in the calc method contains a formula, but the corresponding cell in the field definition contains a database column code, then the records in the Axiom query will be populated with the data from the database column, not the formula from the calc method.

If you are using an in-sheet calc method, then you just need to make sure that single calc method works with the field definition. If you are using a calc method library, then you must make sure that all calc methods that will be used in the query work with the field definition.

Considerations for multiple-row / multiple-column calc methods

You can use multiple-row calc methods with vertical Axiom queries, and multiple-column calc methods with horizontal Axiom queries. The only limitation is that the number of rows or columns in the calc method must be equal to or greater than the number of rows or columns in the field definition. For example:

- You could use a single-row field definition together with a two-row calc method. For each single record of data queried from the database, two rows would be brought into the sheet. The second row of the calc method could be just for formatting purposes (a "spacer" row between data records), or it could contain formulas that reference the data brought in on the first row.
- You could use a multiple-row field definition—for example, one row to bring in data from this year, and a second row to bring in data from last year. In this case the calc method can have two or more rows, such as an additional row for variance calculations (comparing last year to this year), and an additional row to use as a "spacer" row between each block of data.

When the calc method is an in-sheet calc method, it is a relatively simple matter to imagine the field definition layered over the calc method, and ensure that each row (or column) in both items corresponds logically. However, if you are using a calc method library, then you must consider how each possible calc method used in the query will work with the field definition.

The calc methods in the library do not have to have the same number of rows, but they all need to have the same data requirements if they will be used in the same Axiom query, because the data will all come from the same field definition.

For example, imagine that the field definition for the Axiom query is a single row, and the calc method library has two calc methods—one with a single row, and the other with three rows. In both cases, the first row of each calc method will be populated with the data from the single-row field definition. For the three-row calc method, the other two rows will be brought in as well, but no data from the database will come into those rows—which is fine if those two rows are designed for additional data inputs, calculations, or formatting. However, if the three-row calc method is designed to use queried data in more than one row, then it could not be used in the same Axiom query as the single-row calc method, because there is no way to bring in one row of data for one calc method, and multiple rows of data for a different calc method (within the same query).

Formatting considerations

When a calc method is inserted into a sheet, only the formatting properties set for the entire cell are applied. Any formats applied to individual characters in the cell are ignored. For example, if the cell font is set to red or bold in the calc method, then that format will be applied to the cell when the calc method is

inserted into the sheet. But if the cell in the calc method contains text and only one word of that text is set to red or bold, then that formatting is ignored when the calc method is inserted into the sheet, and instead the cell-level formatting is applied.

Using calc method libraries with Axiom queries

When you are creating an Axiom query for use in a plan file (or any other file that supports use of calc method libraries), you can use the sheet's calc method library to apply formatting and formulas to the data rows, instead of using an in-sheet calc method. This approach is typically used to build out the planning rows of plan files. Users can then complete data inputs, add rows as necessary, and even change the calc method used on a row if desired.

To use an Axiom query in a plan file, you first create it in the template. Typically the query would be configured so that when plan files are built from templates, Axiom queries data for the applicable plan code, applies calc methods from the library, and populates the file with data. Usually the template itself does not contain many "fixed" rows of data, so the Axiom queries are used to dynamically insert the accounts or employees relative to the particular plan code for each plan file.

When using a calc method library, each record of data can be assigned a different calc method. For example, some accounts might use an "Input Monthly" calc method for budgeting, while others might use a "Spread Total" calc method. Different formatting and calculations are applied to different accounts as needed.

Each Axiom query in a template can use different calc method settings. For example, you could have one Axiom query that uses an in-sheet calc method to populate the top section of the sheet, and another Axiom query that uses the calc method library to populate the bottom section of the sheet.

The following requirements apply to using calc method libraries with Axiom queries:

- Calc method libraries only apply to templates and plan files. Report files and driver files do not have access to a calc method library, and therefore must use an in-sheet calc method for Axiom queries.
- Calc method libraries can only be used with standard, vertical Axiom queries. Calc method libraries are row-based, and therefore cannot be used with horizontal Axiom queries. If you want to use a horizontal Axiom query in a template or a plan file, then you must use an in-sheet calc method with that query.

IMPORTANT: In order to use a calc method library, the in-sheet calc method setting for the Axiom query must be blank. If an in-sheet calc method is specified for the Axiom query, then that setting will take priority over any other calc method settings for the query.

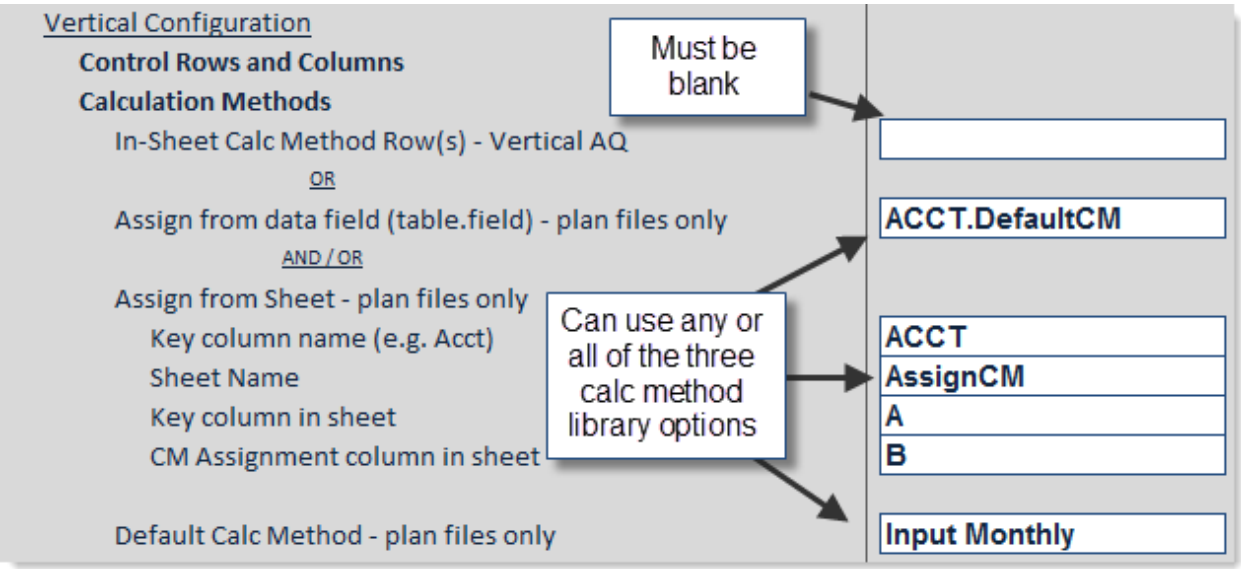
Calc method assignment options

In order to use a calc method library with an Axiom query, Axiom must have a way of determining which calc method to apply to each record in the query. There are several ways to configure this lookup.

- **Look up assignments from a database table**: You can configure the Axiom query to look up calc method assignments from a table in the database. There are two ways to do this:
  - **Single assignment column**: The simplest and most common approach is to set up a single assignment column in the appropriate database table (for example, the ACCT table), and then point the Axiom query to that column to look up the assignments.
  - **Multiple assignment columns by plan code groupings**: If calc method assignments need to vary by a plan code grouping, such as the type of department, then you can use multiple assignment columns. In this case, you create multiple assignment columns in the appropriate database table (such as the ACCT table). Then in the plan code table, you create a "master" assignment column that maps each plan code to its appropriate assignment column in the ACCT table.
- **Read assignments from a sheet**: You can configure the Axiom query to read calc method assignments from a sheet in the file, instead of (or in addition to) looking them up from a database column. This approach requires the most manual setup, but it is the most flexible and can be used for virtually any circumstance.
- **Default calc method**: You can use the default calc method on its own, or in conjunction with the other options. If a calc method's assignment is left blank using any of the options described above, then the default calc method will be used for that record. If desired, you can specify *only* the default calc method (leaving all other calc method options blank), and then the default calc method will be used for all records in the query.

Multiple calc method assignment options can be used in the same query. For example, you can use the sheet option, the database table option, and the default calc method option together. For each record of data in the query, the query first checks the sheet, then checks the database column, and lastly uses the default calc method.

All calc method library assignment options are specified on the Control Sheet for the Axiom query. You must edit the Control Sheet directly in order to use the calc method library, because calc method library settings do not display in the Sheet Assistant. The options are located in the **Vertical Configuration** section of the Axiom query, in the **Calculation Methods** subsection. To use any of these options, the **In-Sheet Calc Method Row(s)** setting must be left blank.



NOTES:

- If a calc method assignment is invalid, then an error results and the Axiom query stops processing. For example, if you specify "InputMonthly" as the calc method assignment, but the actual name of the calc method in the library is "Input Monthly," then no match is found and the assignment is invalid. The default calc method is not used in this case.
- When drilling down an Axiom query that uses the calc method library, the default calc method is used for the drill results, regardless of whichever calc method was used on the row being drilled. If no default calc method is specified for the query, then no calc method is applied to the drill results. This means that the drill results will only present the raw data resulting from the field definition—no formatting and formulas from the row will be carried over to the drill sheet.

Using a single assignment column

To use a single calc method assignment column for an Axiom query:

1. In the appropriate database table, create a column to contain the calc method assignments for each item in the table.

   For example, if the Axiom query is being populated with accounts, you would create a column in the ACCT table and then enter valid calc method names into that column. The column can be named anything you like (for example, AssignCM).

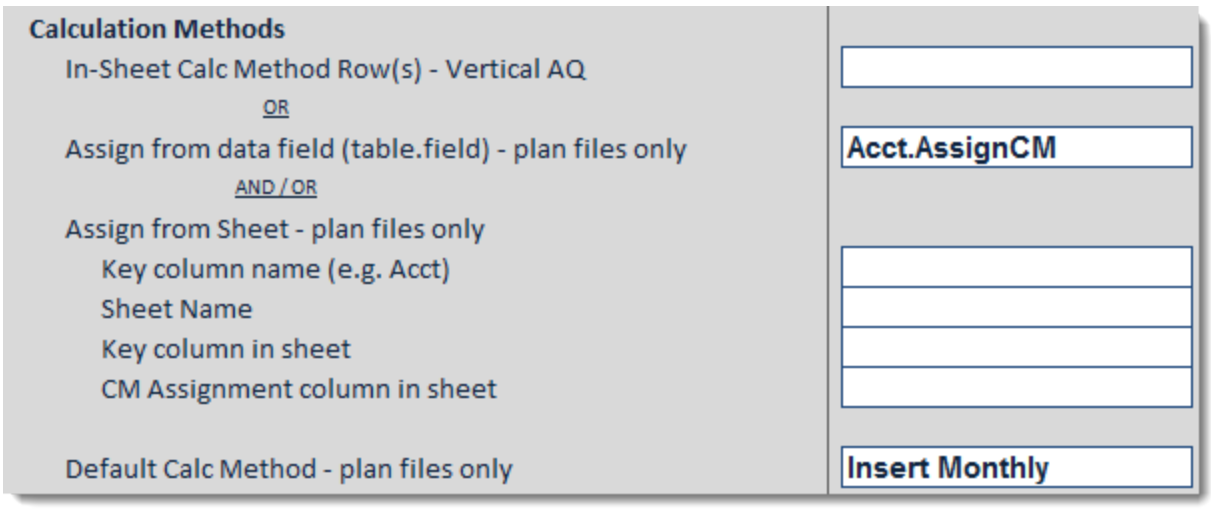
   You can use the same assignment column for multiple Axiom queries if appropriate.

NOTE: If all records in the table need to use the same calc method for this Axiom query, then you do not need to create an assignment column; you can use the default calc method setting instead.

2. Complete the following settings in the Control Sheet for the Axiom query. These settings are located in the **Vertical Configuration** section of the query, under the heading **Calculation Methods**.

Item	Description
Assign from data field	Enter the name of the column that contains the relevant calc method assignments for this Axiom query, using fully qualified Table.Column syntax. For example, `ACCT.AssignCM`. When data records are inserted into a data range, Axiom looks to this column to determine which calc method to use for each record.
Default Calc Method	Optional. Enter the name of a calc method to be used if an entry in the assignment column is blank. If an entry in the assignment column is invalid, then the default calc method is *not* used; an error results instead.

The following screenshot shows an example configuration using a single calc method assignment column:



Using multiple assignment columns by plan code type

The basic implementation of the calc method assignment column assumes that all plan codes (for example, departments) use the same calc method for any single account. If necessary, you can configure the Axiom query to use different assignment columns depending on the type of plan code.

For example, most plan codes might plan for supplies by inputting a total annual value that is spread evenly over months. But for other plan codes, it might be more appropriate to calculate supplies based on a monthly statistic. You can configure the Axiom query so that all plan codes in the first category use a Spread Total calc method for the supplies account, whereas the plan codes in the second category use a Monthly Statistic calc method.

To use different calc method assignments by plan code:

1. Create multiple assignment columns in the appropriate table (for example, ACCT), and complete the columns with the calc method assignments appropriate for a particular type of plan code. The columns can be named anything you like.

   For example, you might have one column that contains the assignments for revenue departments (CMRevenue), and one column that contains the assignments for overhead departments (CMOverhead).

2. In the plan code table (for example, DEPT), create a "master" assignment column. The column can be named anything you like (for example, MasterAssignCM). In this column, for each department, enter the name of the appropriate assignment column from the ACCT table.

   For example, if the department is a revenue department, enter CMRevenue into the MasterAssignCM column.

   Every code in the plan code table must have an entry in the master assignment column (unless you are not creating a plan file for that code, or you know that the plan file for that code does not use this Axiom query).

3. In the Control Sheet settings for the Axiom query, in the **Assign from data field** box, enter the column name for the master assignment column, using fully qualified Table.Column syntax. Then, in parentheses, enter the name of the table containing the detailed assignment columns. The syntax is as follows:

   ```
   MasterTable.MasterAssignCM(DetailTable)
   ```

   For example, if the master column is in the DEPT table and is named MasterAssignCM, and the detailed columns are in the ACCT table, then you would enter the following:

   ```
   DEPT.MasterAssignCM(ACCT)
   ```

   When data records are to be inserted into a data range, Axiom first looks to the MasterAssignCM column in the DEPT table, and finds the entry for the current department. Axiom then finds the specified column in the ACCT table and uses that column to assign calc methods to accounts.

You can optionally complete the **Default Calc Method** setting. If an entry in the assignment column is blank for a particular account, the default calc method will be used. If an entry in the assignment column is invalid, then the default calc method is *not* used; an error results instead.

Reading calc method assignments from a sheet

Using this option, you set up a sheet in the file to contain the calc method assignments, and then you configure the Axiom query to read the assignments from the sheet.

You can use any methodology to create the list of assignments in the sheet. You can manually type in a list of assignments, or you can use an Axiom query to bring in a list that is stored somewhere in the database. If you set up the Axiom query to be dynamic based on criteria such as the current plan code, you can retrieve a different list of assignments for each plan code. This is more flexible than pointing the Axiom query at the database directly using the **Assign from data field** option, because you can store the assignments anywhere in the database, and use any criteria to retrieve them.

To read calc method assignments from a sheet:

1. Set up a sheet in the file to contain the calc method assignments. Within the sheet, one column must contain the relevant key codes (such as ACCT), and one column must contain valid calc method names.

   The sheet can be hidden or visible. Within the target columns, only rows that contain valid key codes will be evaluated for calc method assignments; content such as Axiom query field definitions and column headers will be ignored.

   IMPORTANT: If you are using an Axiom query to retrieve the list of assignments, you must make sure that query runs before the query that will use the assignments. For example, you might set the query to **Refresh on Open**. If the query does not need to refresh on open, or if both queries need to refresh on open, then make sure that the sheet for the assignment query is located before (to the left of) the sheet for the query using the assignments (on the Control Sheet, not the literal sheet order). If both queries are on the same sheet, then the assignment query must have a higher AQ number than the query using the assignments.

2. Complete the following **Assign from sheet** settings in the Control Sheet for the Axiom query. These settings are located in the **Vertical Configuration** section of the query, under the heading **Calculation Methods**.

Item	Description
Key column name	The name of the key column for the calc method assignments. For example, if data is being brought into the sheet by accounts, then enter ACCT.
Sheet Name	The name of the sheet that holds the calc method assignments. For example, you may have created a sheet named CMAssign to hold the assignments.

Item	Description
Key column in sheet	The column in the sheet that contains the key codes. For example, if the key column is ACCT, and the account codes are located in column C of the CMAssign sheet, then you would enter C.
CM Assignment column in sheet	The column in the sheet that contains the calc method assignments. For example, if assignments are in column D of the CMAssign sheet, you would enter D.

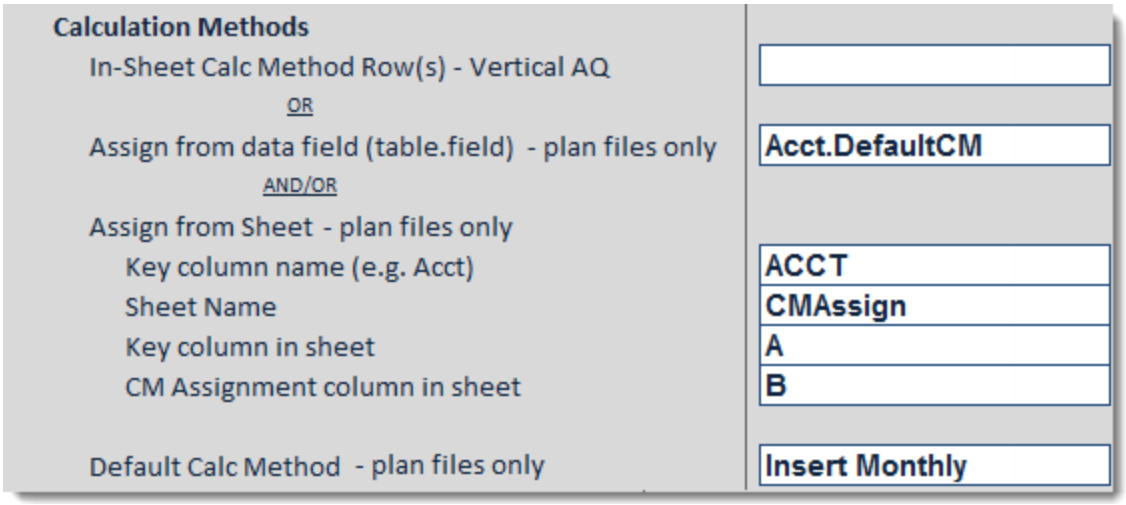
NOTE: If you do not see these settings in your Control Sheet, then you must update the Control Sheet. You can do this using the Sheet Assistant.

All four settings are required in order to read assignments from the sheet. If you specify the key column name but leave any other settings blank, an error will result when the query is run. If you specify any other option but leave the key column blank, then all settings in this section are ignored.

You can also complete the following optional settings:

- **Default Calc Method**: If an entry in the sheet is blank for a particular account, or if a particular account is not listed in the sheet at all, then the default calc method will be used for that account. If an entry in the sheet is invalid, then the default calc method is *not* used; an error results instead.
- **Assign from data field**: If both the **Assign from sheet** and **Assign from data field** options are completed, then Axiom will look up the calc method assignment for each account as follows:
    - Axiom first checks the specified sheet for the account. If a match is found, that assignment is used.
    - If no match is found on the sheet, Axiom checks the specified table column and uses the assignment there. If the assignment is blank in that column, the default calc method is used.

The following screenshot shows an example configuration using the assign from sheet option:



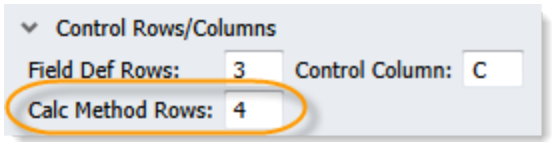
Using in-sheet calc methods with Axiom queries

In-sheet calc methods are used to apply formatting and formulas to data that is inserted into a sheet using an Axiom query. For report files, the in-sheet calc method is the only way to apply formatting and formulas to inserted data. For file group files, the in-sheet calc method can be used as an alternative to using a calc method library.

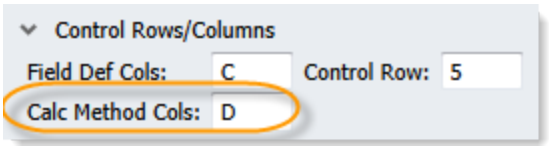
Specifying the location of the in-sheet calc method

The location of the calc method is defined on the file's Control Sheet. If the query is a standard vertical query, you specify one or more calc method rows. If the query is a horizontal query, you specify one or more calc method columns.

You can use the Sheet Assistant to define these settings, or you can manually edit the Control Sheet. The Sheet Assistant updates dynamically depending on whether the query is vertical or horizontal:

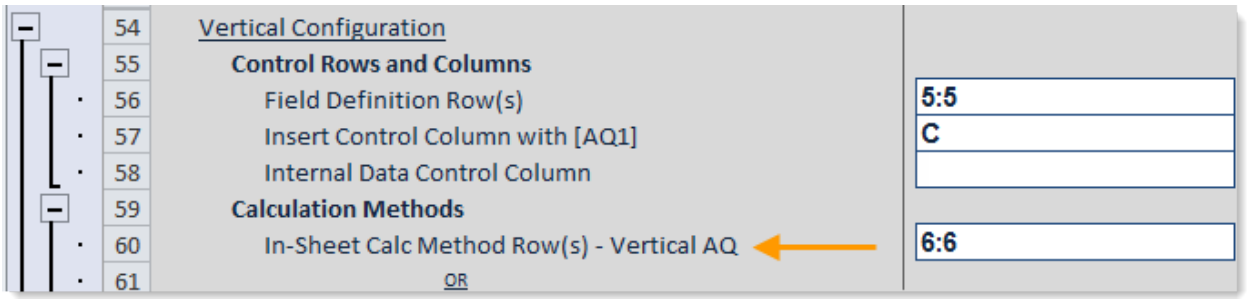


Vertical query

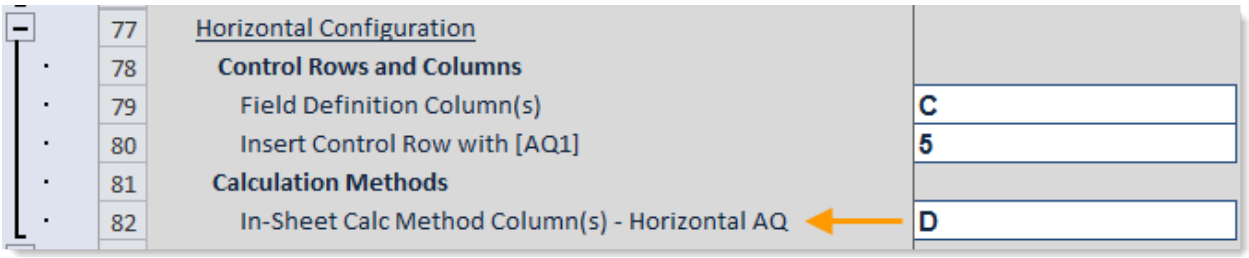


Horizontal query

In the Control Sheet, you must expand the appropriate section to reach the setting, either **Vertical Configuration** or **Horizontal Configuration**.



Vertical configuration



Horizontal configuration

Keep in mind that once you have specified a row or a column location, it does not update automatically if you insert rows or columns into the sheet, or if you move your calc method in the sheet. You may want to first design the query on the sheet, figuring out where you want to place items such as headers and subtotals (if needed), and then complete the query settings. If you do change the location in the sheet, you must update the Control Sheet setting.

Creating the in-sheet calc method

Within the specified rows or columns of the in-sheet calc method, define any formulas and formatting that you want applied to data records as they are inserted into the data range of the Axiom query. For example, you might:

- Format cells that will contain financial data with the desired number or currency formats.
- Add variance calculations or other formulas to be applied to the data.

The in-sheet calc method and the field definition must complement each other. As you are constructing both, imagine the field definition layered on top of the in-sheet calc method. The combination of both is how the data records will appear when the file is refreshed. If there is a conflict, the field definition takes priority. For example, if a cell in the in-sheet calc method contains a formula, but the corresponding cell in the field definition contains a database column code, then the records in the data range will be populated with the data from the database column, not the formula from the calc method.

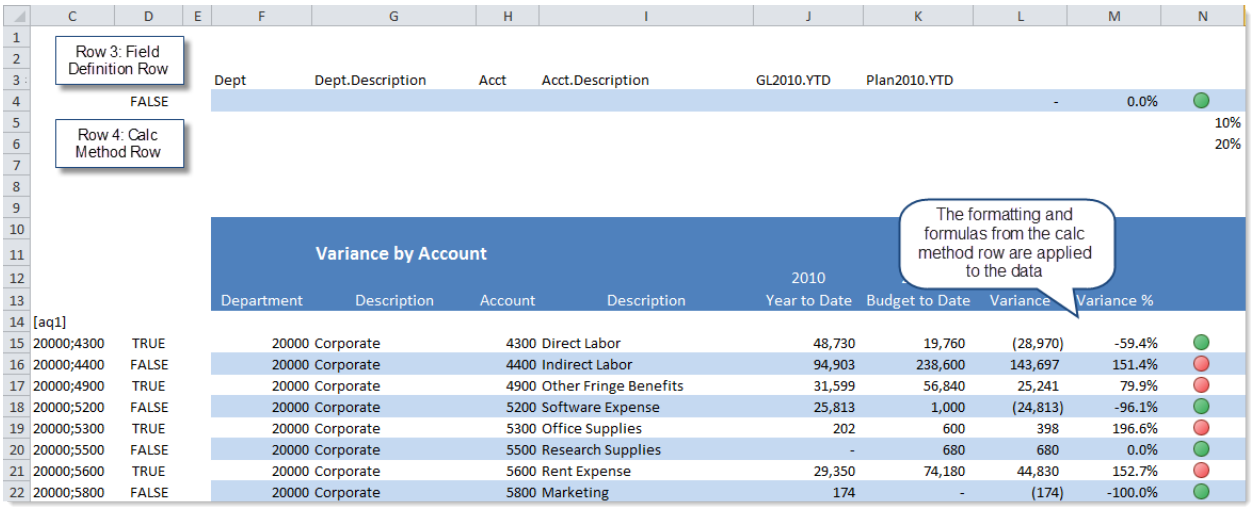
The number of rows or columns in the in-sheet calc method must be equal to or greater than the number of rows or columns in the field definition. For example, if the field definition is two rows, then the in-sheet calc method can be two rows, or three rows, but it cannot be one row. If the in-sheet calc method is smaller than the field definition, it will be ignored and no calc method will be applied to the query when it is run.

NOTES:

- The calc method is only applied when data is inserted into a data range; it is not applied when existing data is updated. If the refresh behavior for an Axiom query is rebuild, then the calc method is always applied because existing data is deleted and new data is inserted. If the refresh behavior is update and insert, then the calc method is only applied to any new records that are inserted; the calc method is not applied to existing data records that are updated.
- The calc method's row height or column width is only applied to the data range when the insertion behavior is **Insert**. If the behavior is **InsertRange**, then existing row height and column width is left as is, regardless of the calc method settings.

Example in-sheet calc method

The following example screenshot shows how an in-sheet calc method was applied to the data in the data range. In addition to cell formatting and formulas, this calc method also uses Data Validation to alternate the cell shading for each row, and to flag each row with a green light or a red light, depending on whether the row's variance is within acceptable ranges.



	C	D	E	F	G	H	I	J	K	L	M	N
3				Dept	Dept.Description	Acct	Acct.Description	GL2010.YTD	Plan2010.YTD			
4		FALSE								-	0.0%	
5												10%
6												20%
11				Variance by Account								
12								2010				
13				Department	Description	Account	Description	Year to Date	Budget to Date	Variance	Variance %	
14	[aq1]											
15	20000;4300	TRUE		20000	Corporate	4300	Direct Labor	48,730	19,760	(28,970)	-59.4%	
16	20000;4400	FALSE		20000	Corporate	4400	Indirect Labor	94,903	238,600	143,697	151.4%	
17	20000;4900	TRUE		20000	Corporate	4900	Other Fringe Benefits	31,599	56,840	25,241	79.9%	
18	20000;5200	FALSE		20000	Corporate	5200	Software Expense	25,813	1,000	(24,813)	-96.1%	
19	20000;5300	TRUE		20000	Corporate	5300	Office Supplies	202	600	398	196.6%	
20	20000;5500	FALSE		20000	Corporate	5500	Research Supplies	-	680	680	0.0%	
21	20000;5600	TRUE		20000	Corporate	5600	Rent Expense	29,350	74,180	44,830	152.7%	
22	20000;5800	FALSE		20000	Corporate	5800	Marketing	174	-	(174)	-100.0%	

Refresh and insertion behavior for Axiom queries

When an Axiom query is refreshed, data is updated according to the settings of the query. Axiom queries support several options to control:

- How data is updated (rebuild, update, insert)
- How data is inserted when using rebuild or insert
- What actions cause the Axiom query to run (manual refresh, document processing, file open, etc.)

Controlling when an Axiom query is refreshed

Axiom queries support several options to control when an Axiom query is refreshed. By default, Axiom queries will refresh in the following situations:

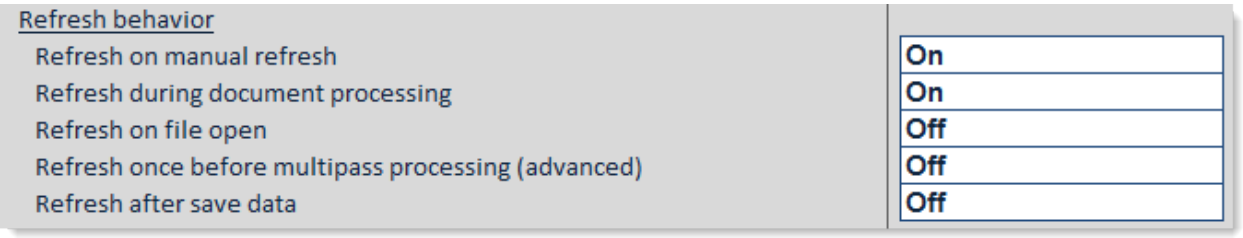
- When a user manually refreshes the sheet (such as by clicking the **Refresh** button in the ribbon).
- When the file is processed by a feature that includes performing a refresh, such as Process Plan Files, File Processing, or Process Document List.

If desired, you can disable these options so that a refresh does not occur in these circumstances, and/or you can enable other refresh options, such as automatically refreshing on open, or refreshing before multipass processing occurs.

NOTE: In all circumstances, Axiom queries will only refresh if they are active. If a query is not active, then it is essentially ignored by all processes.

Configuring refresh options for an Axiom query

Refresh options are located in the **Refresh behavior** section of the Control Sheet:



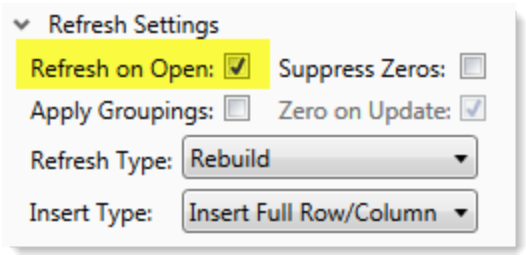
The following options are available:

Item	Description
Refresh on manual refresh	If enabled, the query will be refreshed when a user manually refreshes the sheet. For example, the user could click the **Refresh** button in the ribbon, or click a task pane item that is configured to perform a refresh, or press F9. This option is enabled by default. If you disable this option, then the query will not refresh when a user manually refreshes the sheet. For example, you might disable this option if you have a query that you *only* want to run when the file is opened, and not during any subsequent manual refreshes. **NOTE:** In plan files only, users cannot manually refresh Axiom queries unless they have the **Run AQs in Plan Files** permission. If you have a query that needs to be run in plan files, but you do not want to give users this permission, then you can enable **Refresh on file open** for the query.

Item	Description
Refresh during document processing	If enabled, the query will be refreshed when the file is processed by a feature that includes performing a refresh, such as Process Plan Files, File Processing, or Process Document List. Some of these features, such as Process Plan Files, allow you to disable refreshing particular queries as part of that particular processing job. For other features, the query will always be refreshed if this option is enabled. This option is enabled by default. If you disable this option, then the query will not be available to processing features that include performing a refresh.
Refresh during template processing	If enabled, the query will be refreshed when the file is processed by the Scheduler task Process Template List. This option only applies to file group template files; it has no effect in any other file type. When the query is processed by Process Template List, the **Last refresh time** of the query is updated in the template. This feature is intended to enable use of time-stamped Axiom queries in virtual plan files, by time-stamping the query in the template.
Refresh on file open	If enabled, the query will be refreshed automatically whenever the file is opened, whether by a user or by a system process (with one exception: Process Plan Files—see notes below). This setting is disabled by default. When this setting is enabled, administrators can use the **Open Without Refresh** option if you need to open the file without running the Axiom query. For more information on this option, see Opening an Axiom file without refreshing data. **NOTES:** • If the file uses refresh forms (refresh variables or Axiom form as refresh form), the refresh form will only display on file open if you explicitly configure it to do so using the **Refresh Forms Run Behavior** setting on the Control Sheet. • In plan files, queries configured to refresh on open are always run when the file is opened by a user, regardless of whether the user has the **Run AQs in Plan Files** security permission. This can be used if you have a query that needs to be run to enable functionality in the plan file (such as for a drop-down list), but you do not otherwise want users to be able to refresh Axiom queries. • This setting is ignored by Process Plan Files. However, the query can still be selected for processing if it is also enabled to **Refresh during document processing**.

Item	Description
Refresh once before multipass processing	This option only applies when the file is processed using the multipass option of File Processing. If enabled, the query will be run once before multipass processing begins. If you do not want the query to also be run during each individual pass, then you should disable **Refresh during document processing**. This option is disabled by default. You might enable this option if you have a query that needs to be run to bring an initial set of data into the file, but you don't want to set the query to refresh on open because the query is quite large and causes an unnecessary delay when opening the file for editing.
Refresh after save data	If enabled, the query will be refreshed automatically after a save-to-database occurs, whether the save is initiated by a user or by a system process. The refresh occurs regardless of whether any data changes were actually saved. The Axiom query is refreshed after *all* save-to-database processes in the workbook are completed. This setting is disabled by default. You might enable this option if you have a configuration where the save-to-database process adds or updates a record that would be returned by an Axiom query in the workbook. In order for the data in the Axiom query to be updated automatically for the changed data, you would need to enable this option (otherwise the user would not see the changed data until they manually refresh). This feature may be especially useful in Axiom form design. **NOTE:** The behavior of this option depends on whether the sheet-level setting **Convert Axiom Query results to zero on save** is also enabled. If "zero on save" is enabled, then the Axiom query is first zeroed, then the file is saved, then the Axiom query is refreshed with data. This order is so that the data is not saved within the file. However, if "zero on save" is *not* enabled, then the Axiom query is refreshed before the file is saved, which means that the current data will be saved within the file. This does not apply when interacting with Axiom forms, because in that case the file is never saved.

Most of these refresh options are not available on the Sheet Assistant and must be configured on the Control Sheet if you want to deviate from the default behavior. There is one exception—if you want to configure a query to refresh on open, this setting is available in the **Refresh Settings** section of the Sheet Assistant:



NOTE: If you want an Axiom query to refresh before a particular save-to-database block is processed, this behavior can be configured in the Save2DB tag itself. For more information, see Running an Axiom query before a save-to-database.

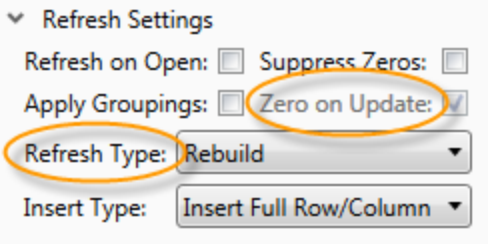
Specifying how data is refreshed in an Axiom query

Axiom queries support several different options that control how the data is updated when an Axiom file is refreshed. The primary decision point is whether to completely rebuild the data ranges each time the query is refreshed, or to use some combination of insert, update, and zero.

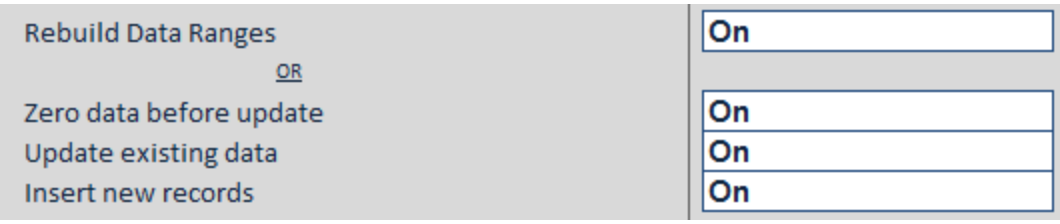
Configuring the refresh type for an Axiom query

You can use the Sheet Assistant to specify the refresh type for the query, or you can modify the Control Sheet directly.

On the Sheet Assistant, you specify the desired refresh type as one of the following: **Rebuild**, **Insert Only**, **Update Only**, or **Insert and Update**. If an update option is selected, then you can specify whether zero is enabled by selecting or clearing the **Zero on Update** check box (by default, it is selected when an update option is selected).



On the **Control Sheet**, each option is individually controlled as an **On** or **Off** selection within the **Refresh behavior** section. If rebuild is set to On, then the settings for insert, update, and zero are ignored.



NOTE: On the Control Sheet, it is possible to set all refresh options to Off. In this case, no action occurs when the query is refreshed. If you do not want a particular query to refresh with data, you should inactivate the query rather than disable all refresh options.

Rebuild behavior

If rebuild is enabled for an Axiom query, then all existing content in data ranges is deleted, and data ranges are rebuilt. This is the default setting for all Axiom queries.

When rows or columns are inserted as part of a rebuild, the **Insertion Behavior** setting for the query determines whether the data ranges are rebuilt by deleting and then reinserting entire rows or columns, or whether only the cells in the specified range are deleted and inserted. For more information, see Insertion options for Axiom queries.

This setting is only used in circumstances where you do not need to retain any inputs or changes to the rows within data ranges (or columns, for horizontal queries). Data ranges are completely rebuilt, using the most current data and the most current versions of calc methods. Generally, rebuild is used for report files and not templates/plan files.

However, if appropriate, rebuild can be used in any context. For example, imagine that you wanted to generate a list of accounts in a plan file, to create a drop-down list for users to select an account number for a newly inserted calc method. In this case you would use rebuild to ensure that the list of accounts matches the current list in the database.

Either the database column sort or the spreadsheet column sort can be used to sort the data in the rebuilt range.

Insert behavior

If insert is enabled for an Axiom query, then any records resulting from the data query that do not currently exist in the data range are inserted into the data range. Queries can be set to insert only, or used in conjunction with update.

When rows or columns are inserted, the **Insertion Behavior** setting for the query determines whether entire rows or columns are inserted, or whether only the cells in the specified range are inserted. For more information, see Insertion options for Axiom queries.

The database sort settings only apply if no data currently exists in the range. If data already exists in the range, then any new rows of data are inserted at the bottom of the range. If spreadsheet sort settings are defined, the data range is sorted after the refresh.

NOTE: If a query with multiple data ranges is set to insert, and a record exists in one data range but does not exist in another data range, then the record will not be inserted in the other data ranges. However, if the record does not exist in any data ranges, then it will be inserted in each eligible data range.

Update behavior

If update is enabled for an Axiom query, then data values in the data ranges that correspond to field definition entries are replaced with new data from the database. Queries can be set to update only, or used in conjunction with insert.

Only matched data is updated. If a data point exists in the data range but does not exist in the query (unmatched data), then the update process ignores that existing data point. If you do not want to retain this unmatched data as is, then you can use the zero option in conjunction with the update.

NOTE: It is strongly recommended to enable zero when using update, to ensure that no stale data displays in the sheet. You should only disable zero if you explicitly need to keep the old unmatched data.

The update process only updates data; it does not apply calc methods. If changes have been made to calc methods (either in the library or to the in-sheet calc method), those changes are not applied to existing data in the data ranges. The only way to apply calc method changes to *existing* data is to perform a rebuild. Note that if insert is enabled in conjunction with update, then any new records inserted will use the most-current versions of calc methods.

The database column sort does not apply to this option; the spreadsheet column sort does apply.

NOTE: The update option uses key codes in the data control column (or row) to match and update each record of data. If data records were originally inserted by using rebuild or insert, then Axiom automatically placed the appropriate key codes for each record. If data records were not inserted by the Axiom query, then you must populate the data control column (or row) with the appropriate key codes in order to facilitate the update. For more information, see Manually placing data control codes for update queries.

Zero on update behavior

If zero is enabled for an Axiom query, then existing data values in the data range that correspond to field definition entries are zeroed before they are updated. This option only applies if update is enabled.

This setting is intended for circumstances where a data point existed in the database when the data range was first populated, but that data point was subsequently deleted from the database. (Or, the data filter for the Axiom query was changed to exclude records that were previously included in the query.) If zero is not enabled, then the original value is retained in the data range because the update option ignores it. When the zero option is enabled, the deleted data value is replaced with a zero.

The following columns are not zeroed:

- Columns used as the Axiom query "sum by" level
- Key columns and alternate key columns
- Validated columns
- Columns from reference tables, when the primary table of the query is a data table

If alternate aggregation (such as RowCount) is used on a column that would normally not be zeroed, then that column will be zeroed—unless the column is on a reference table and the primary table is a data table.

Refresh data only when dependent tables have been modified (time-stamped Axiom queries)

You can configure an Axiom query so that it will only refresh data if one or more dependent tables have been modified since the last execution of the query. If the dependent tables have not been modified, then the query will not execute and the existing data in the sheet will be left as is. This feature is known as *time-stamped Axiom queries*.

This configuration is primarily intended for Axiom queries that bring in supporting data for the file, where this supporting data does not change frequently. Eliminating the automatic execution of these queries when the table data has not changed may improve file performance.

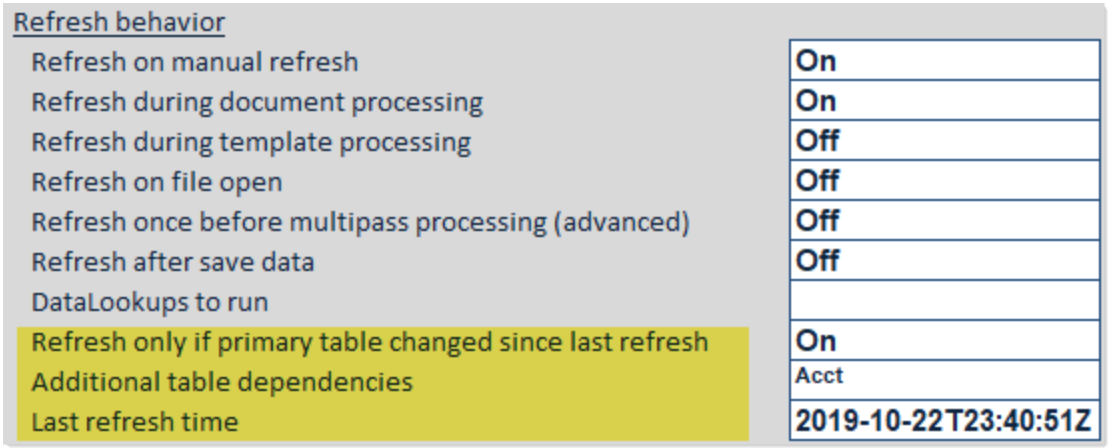
By default, the dependent table is the primary table for the query. You can also optionally specify additional dependent tables when enabling the time-stamped refresh behavior.

Configuring a query as a time-stamped query

To configure an Axiom query as a time-stamped query, use the following settings. These settings can only be viewed and modified on the Control Sheet, in the **Refresh behavior** section of the Axiom query settings.

Item	Description
Refresh only if primary table changed since last refresh	Specifies whether the query uses time-stamped refresh behavior: • If **On**, then the query will only run if dependent tables have been modified since the last time the query was refreshed. The dependent tables for the query are the primary table and any additional tables listed in the **Additional table dependencies** field. The query must also meet the following requirements: ◦ The primary table of the query must be explicitly defined in the query settings, not inferred from field definition entries. ◦ The sheet-wide setting **Convert Axiom Query results to zero on save** must be disabled for the sheet. If it is enabled, then the **Refresh only if primary table changed since last refresh** setting will be ignored, and the query will zero and refresh as normal. • If **Off** (default), then time-stamped behavior does not apply and the query will run whenever it is eligible to refresh.

Item	Description
Additional table dependencies	Optional. One or more additional tables to consider when using time-stamped refresh behavior. Only applies if **Refresh only if primary table changed since last refresh** is enabled. If a table is listed here, then the query will run if that table has been modified since the last time the query was refreshed (regardless of whether the primary table has been modified). You can list multiple table names, separated by commas. It is not necessary to list the primary table here, as the primary table will always be considered.
Last refresh time	The date and time that the query was last run. The last modified date-times of the dependent tables are compared to this date-time to determine whether the query should be run. The last refresh time is automatically stamped in this field when the query is run. This date-time is written in UTC (Coordinated Universal Time) to ensure consistent date-time comparisons—the date-time is *not* converted to the local time of the client where the refresh occurred. See the following section, *How time-stamped queries work*, for more information.



Example Axiom query with time-stamped behavior enabled

How time-stamped queries work

Time-stamped queries use the **Last refresh time** for the query on the Control Sheet. Each time an Axiom query is run, the date and time of the refresh is written to this field. If **Refresh only if primary table changed since last refresh** is enabled, then before the query is executed, Axiom compares the last refresh date-time of the query to the last modified date-times of the dependent tables:

- If no dependent tables have been modified since the last refresh of the Axiom query, then the query is not refreshed and the existing data in the sheet is left as is.

- If any of the dependent tables have been modified since the last refresh of the Axiom query, then the query is refreshed.
- The dependent tables for the query are the primary table and any tables listed in the **Additional table dependencies** field.

The **Last refresh time** field is system-controlled and should not be manually modified. The only exception is that you can clear the field if you want to "reset" the last refresh time of the query. If needed, you can clear the field manually, or you can use the **Clean Document** tool on the Axiom Designer ribbon tab.

NOTES:

- The last modified date-time of tables reflects both data changes and structure changes. The query will also refresh if the structure or properties of any dependent tables have changed.
- By default, the **Last refresh time** field is not populated when refreshing Axiom queries in a template file for a file group. This is because when you create plan files from the template, the field must be blank so that it can be populated for each individual plan file. However, if you are using virtual plan files and therefore you want to maintain a time stamp in the template itself, you can use the **Process Template List** task for this purpose. This task runs Axiom queries set to **Refresh during template processing** and then time-stamps them for use in virtual plan files. For more information, see the *Scheduler Guide*.

The time-stamped refresh behavior is honored by all Axiom query refresh settings (manual refresh, file open, after save data, etc.). If you enable this feature for a query, and then you later change something about the query—for example, if you add a column to the field definition, or you change a formula in the in-sheet calc method—then you should clear the last refresh time of the query so that the query will execute using the new query settings. Otherwise the query will not execute again until a dependent table is modified.

Design considerations

Generally speaking, the time-stamped query feature is only a good fit for queries that meet the following criteria:

- The Axiom query settings are static. Query settings do not use formulas to dynamically change based on other factors such as refresh variables. The exception would be an Axiom query in a plan file that uses a formula to filter the query based on the current plan code, because in that case the filter will always be the same for that plan file.

  For example, you would not want to use this feature with a query where the filter changes based on a refresh variable selection, because the query would not refresh to reflect the changed filter.

- The data in the dependent tables is mostly static. Modifications to the table data are rare. If the query has to run each time the file is opened because the dependent tables keep getting modified, then this configuration does not provide much benefit.

- The data in the dependent tables is available to all users who will access the file. The users do not have security filters that restrict access to the tables. If the users who access the file have different security filters on the dependent tables, then the query needs to always run to ensure that the current user's filter is applied to the query.

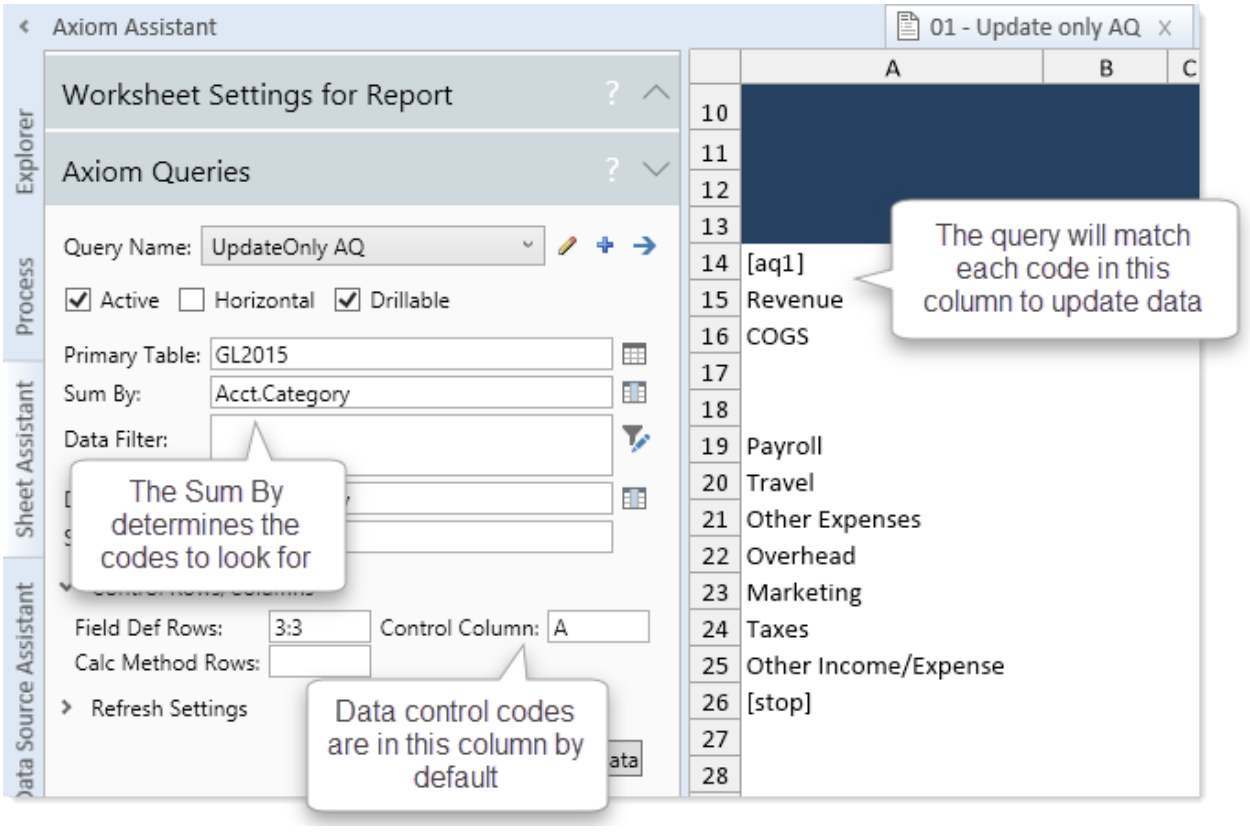
If you enable this feature for a query that does not meet these criteria, the setting may not have any benefit and/or the query results may not be as expected.

NOTE: This feature is not supported for use with file processing. Specifically, it should not be enabled for any queries that are executed for each pass of file processing. It could be enabled for a query that runs on file open to provide supporting information for the file, but otherwise the query is not executed during file processing.

Manually placing data control codes for update queries

When the refresh behavior of an Axiom query is set to **Update**, Axiom references the data control codes in the insert control column to determine the data to update in each row.

In the following example, the sum by for the query is `Acct.Category` and the insert control column is column A. When the query is refreshed, Axiom checks column A for valid Acct.Category codes and then updates the row with the appropriate data for that code.



When data is inserted by an Axiom query (using either Rebuild or Insert), the query automatically adds the appropriate codes to this column. You could then change the behavior of the query to Update and the codes will already be there for subsequent updates.

However, in some cases you want to design an Axiom query to use Update refresh behavior in files where the codes are not already placed in the sheet. This means that you must manually add the codes to the appropriate column. For example:

- Some plan files are designed so that updated data is periodically brought into the files by running an Axiom query. If users are allowed to add rows to these files by inserting calc methods, then the calc methods must be designed to place their relevant key codes in the insert control column, or else those rows will not be updated when the Axiom query is run.
- In a report, you may want to create a "static" income statement by manually setting up each row, and then using an update-only Axiom query to bring in the latest data. In this case you must manually place the key codes in the insert control column to use for the update.

The data control codes placed in the sheet must correspond to the "sum by" level for the query. If multiple columns are used for the sum by, then multiple codes are separated with semicolons. The codes must be placed in the same order that the columns are listed in the "sum by" field.

For example, if the sum by level for the query is `ACCT.ACCT;DEPT.DEPT`, then the data control codes for the query would be similar to the following:

```
10000;100
```

Where 10000 is the account represented on the row, and 100 is the department.

NOTE: If the field definition spans multiple rows or columns, then the codes should be placed on each row or column that corresponds to the field definition. For example, if the field definition is three rows, then the insert control column should have three sequential rows for each key code.

Treatment of duplicate codes within a data range

If a data range contains multiple records with the same data control code, then Axiom will treat those duplicate codes as follows:

- If the duplicate codes are not contiguous (meaning there is a different code between the duplicate codes), then Axiom will update each duplicate record with the appropriate data for that code.
- If the duplicate codes are contiguous, then only the first record in the contiguous block will be updated.

In this context, "record" refers to each distinct data record that corresponds with the query's field definition. If the field definition is a single row, then each row is a record and each row will be evaluated individually. If the field definition is multiple rows, then each distinct multi-row block is treated as a record. For example, if the field definition is two rows, then each record consists of two rows with the

same code—those rows are not considered to be contiguous in this case because both rows belong to the same record. However, if those two rows are followed by another two-row record with the same code, then the second record is contiguous and will not be updated.

If duplicate codes are separated by a blank row (a row without a code), then by default Axiom will treat that row as a separator and will update each duplicate record. However, if you want Axiom to ignore blank rows and instead treat the records as contiguous, then a system configuration setting is available to change the behavior system-wide (**AllowBlanksToBeDataRowSeparatorsInAQDataUpdate**).

Insertion options for Axiom queries

The insertion behavior setting for an Axiom query determines how data records are inserted into a data range (and also how they are deleted, if the refresh behavior is rebuild).

Axiom queries have two options for insertion behavior: **Insert Full Row/Column** and **Insert Range**. Insert Full Row/Column is the default setting that will be used for most queries. Insert Range is an advanced option that is primarily intended for horizontal queries, but may also be appropriate for certain vertical query configurations.

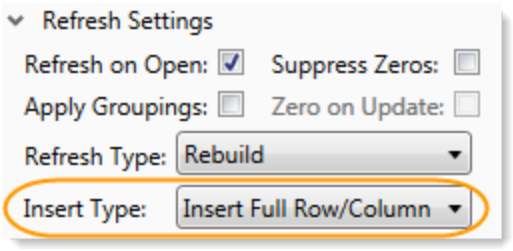
Insertion behavior settings apply if the refresh behavior for the Axiom query is set to either rebuild or insert. If a query is set to update only, then insertion behavior does not apply.

NOTE: If you are configuring a horizontal Axiom query within a template, plan file, or file group utility, then you must use Insert Range if the sheet uses calc methods. Inserting full columns is not supported in this context.

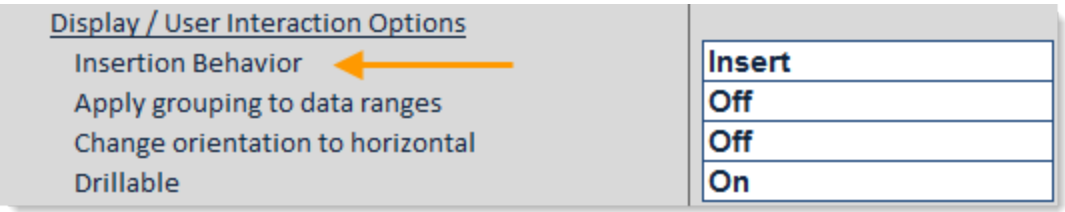
Configuring the insertion behavior

You can use the Sheet Assistant to specify the insertion behavior for the query, or you can modify the Control Sheet directly.

In the Sheet Assistant, the setting is located in the **Refresh Settings** section, as **Insert Type**. The options are **Insert Full Row/Column** and **Insert Range**.



On the Control Sheet, the setting is located in the **Display/User Interaction Options** section, as **Insertion Behavior**. The options are **Insert** and **InsertRange**.



How Insert Full Row/Column works

For standard vertical Axiom queries, if **Insert Full Row/Column** is specified as the insertion behavior, then the Axiom query is populated by inserting new rows within the data ranges (the area flagged by the `[AQ#]` and `[Stop]` tags). Any content below the `[Stop]` tag is pushed down.

If the refresh behavior for the query is rebuild, then all rows within the data range are first deleted, and then new data is inserted. If the refresh behavior is insert and update, then existing rows are retained, but any new data rows are inserted below the existing data.

If the query is a horizontal query, then the same behavior applies, except that columns are inserted and deleted instead of rows. This behavior is often not desirable for a horizontal query, especially if the horizontal query is being used to build out components for a vertical query. If entire columns are inserted or deleted, this may adversely affect settings, formulas, and formats needed for the vertical query. In many cases, horizontal queries will use the Insert Range behavior instead.

NOTE: When the insertion behavior is Insert Full Row/Column, the row height or column width of the calc method (depending on the query orientation) is applied to new records coming into the data range.

How Insert Range works

If **Insert Range** is specified as the insertion behavior, then the Axiom query is populated by inserting new cells within its designated range. This is equivalent to inserting within a spreadsheet and choosing to shift cells down or right, rather than inserting entire rows and columns.

Insert Range is most often used for horizontal queries, when they are being used to build out components for a vertical query. This way you can refresh the horizontal query without affecting any content in the sheet that lies above or below the horizontal data range.

Most vertical queries do not need to use Insert Range, because all necessary content for the data rows is contained within the field definition and calc method, and any additional settings for the query are normally maintained at the top of the sheet, in an area unaffected by row insertion and deletion. However, it may be appropriate for certain vertical query configurations. For example, you may want to create a "portal-style" view on your home page, displaying information side by side. Using Insert Range, you could have a query on the home page that could be updated without impacting the information displayed to either side of the query.

When using Insert Range, you must designate the range to be affected by the cell insertion. This is accomplished by placing `[Startrange]` and `[Stoprange]` tags in the same row or column as the `[AQ#]` tag (depending on the query orientation). For more information, see Defining the insertion range when using Insert Range.

NOTES:

- When the insertion behavior is Insert Range, the row height or column width of the calc method (depending on the query orientation) is not applied to the data range. The existing row heights / column widths are retained.
- When using Insert Range, you must use an in-sheet calc method.

Defining the insertion range when using Insert Range

If the insertion behavior for an Axiom query is **Insert Range**, then you must define the range to be affected by the cell insertion. This is accomplished by placing `[Startrange]` and `[Stoprange]` tags in the same row or column as the `[AQ#]` tag (depending on the query orientation).

- The [Stoprange] tag is required to identify the end of the range. If the [Stoprange] tag is omitted, then the end of the range is the used range of the sheet, which in most cases effectively defeats the purpose of using Insert Range. However, it can be omitted if this is the desired behavior.
- The [Startrange] tag is optional to identify the start of the range. If the [Startrange] tag is omitted, then the start of the range is the [AQ#] tag. In many cases, this is the desired start of the range, so no additional tag is necessary. However, if you need the range to start "earlier," you can place the [Startrange] tag to the left or above the [AQ#] tag (depending on the query orientation). This may be necessary for certain advanced configurations such as nested queries, where you need the range to extend to include the tags of the "parent" query.

These tags specify the insertion range, and define the starting point and ending point of the field definition and the calc method. Cells are only inserted within this boundary.

The [Stoprange] and [Startrange] tags will only be honored if they are placed in the correct row or column, and in the correct locations.

- If the query is a vertical query, the [Startrange] tag can only be to the left of the [AQ#] tag, and the [Stoprange] tag can only be to the right of the [AQ#] tag, all within the same row.
- If the query is a horizontal query, the [Startrange] tag can only be above the [AQ#] tag, and the [Stoprange] tag can only be below the [AQ#] tag, all within the same column.

If the tags are in invalid locations, then they will be ignored, and the insertion range is determined as if the tags were omitted.

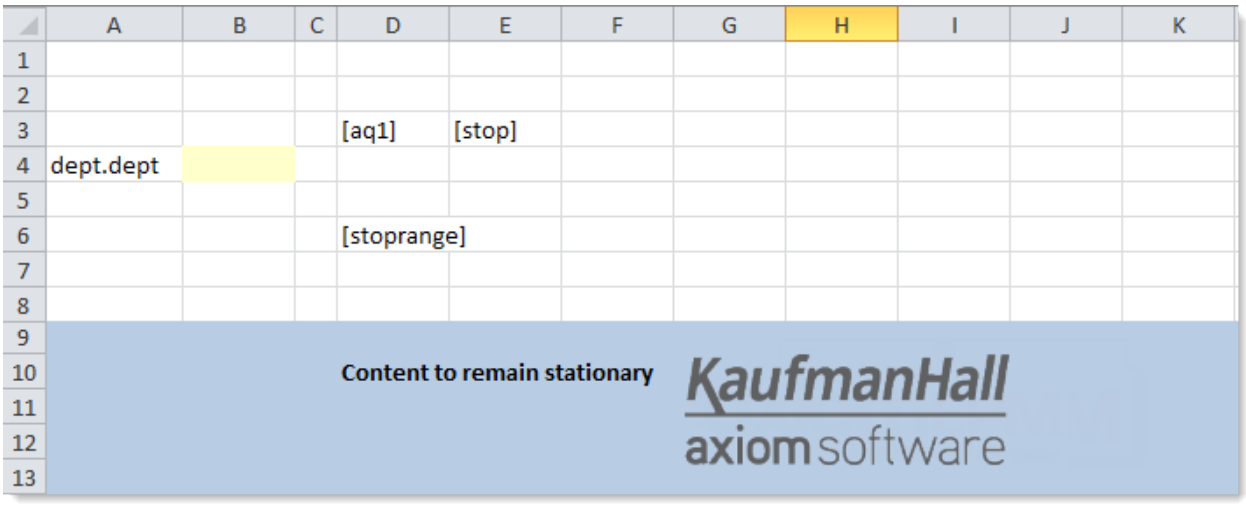
NOTES:

- The [Stoprange] and [Startrange] tags only apply when the insertion behavior is Insert Range. If the behavior is Insert Full Row/Column, the tags are ignored.
- For vertical queries, Insert Range is not supported for use with calc method libraries. If the [Startrange] and [Stoprange] tags are used in this circumstance, then the field definition will honor the range but the calc method will ignore it.

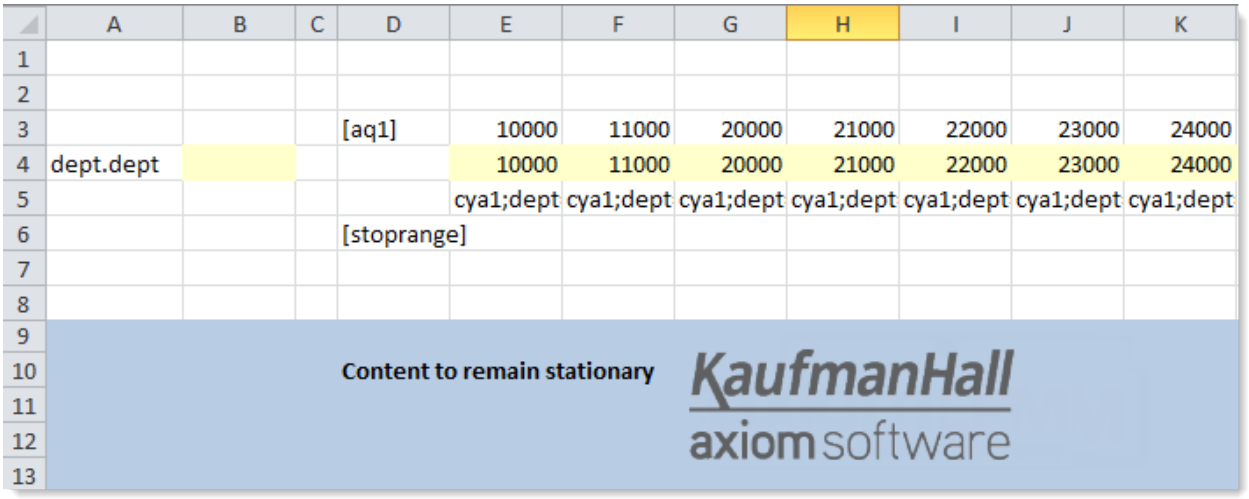
Horizontal queries

If the query is a horizontal query, then the [Stoprange] tag is placed below the [AQ#] tag, within the same column, to specify the outer limits of the field definition and calc method. When the query is run, anything above the [AQ#] tag and below the [Stoprange] tag will be unaffected. If needed, you can place the [Startrange] tag *above* the [AQ#] tag, within the same column.

The following example shows a horizontal Axiom query (before refresh):



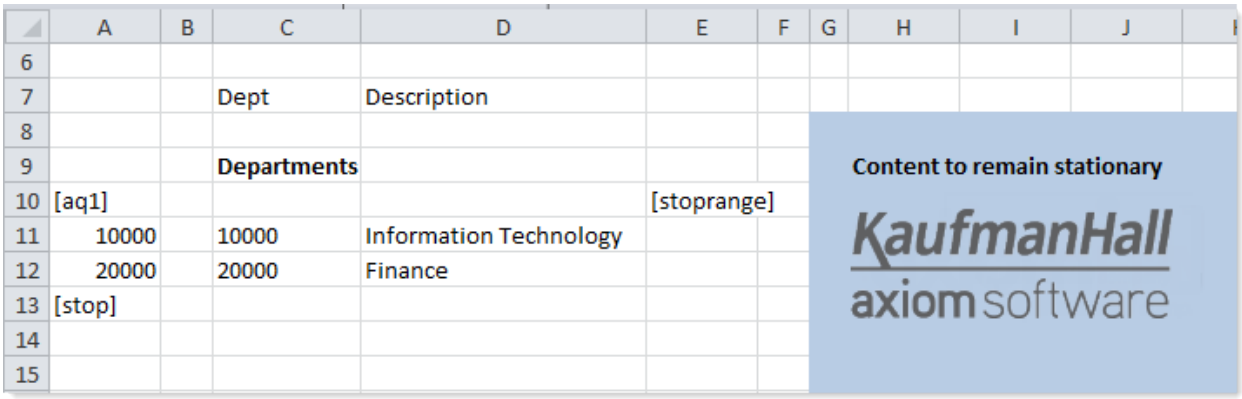
When the query is refreshed, the horizontal query is populated by inserting cells instead of inserting columns. The areas above the [AQ#] tag and below the [Stoprange] tag are unaffected.



Vertical queries

If the query is a vertical query, then the [Stoprange] tag is placed to the right of the [AQ#] tag, within the same row, to specify the outer limits of the field definition and calc method. When the query is run, anything to the left of the [AQ#] tag and to the right of the [Stoprange] tag will be unaffected. If needed, you can place the [Startrange] tag to the *left* of the AQ tag, within the same column.

The following example shows how a vertical Axiom query could be set up to use Insert Range. This configuration might be used for a "portal style" sheet, such as on the Axiom home page or on a Summary or Instructions sheet in a plan file. This is example is simply to show how the tags are placed—for a real implementation you would likely use some combination of freeze panes and views to hide the setup information.

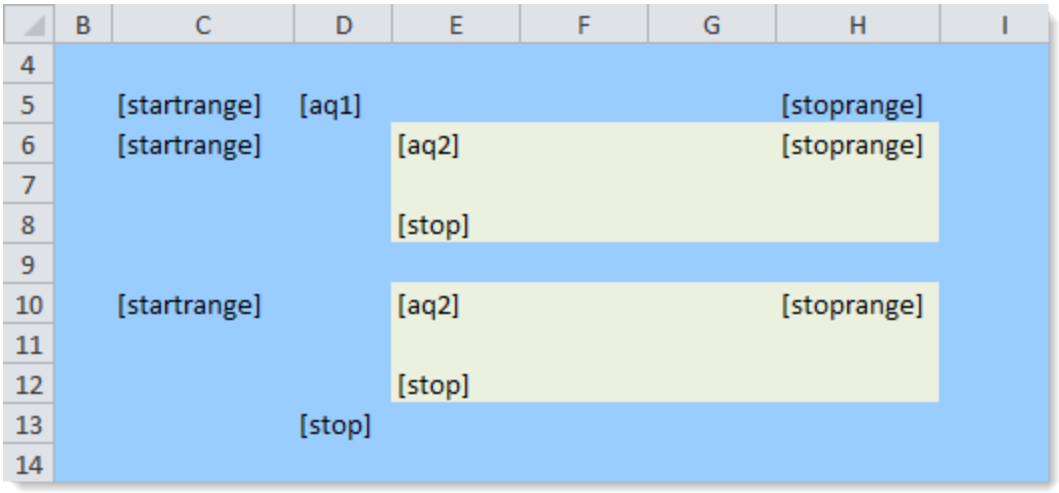


Startrange example

For certain advanced configurations, the [Startrange] tag can be used to extend the affected range past the [AQ#] tag.

For example, if you have two nested Axiom queries, the tags of both queries must be kept in sync. If [Startrange] is not used, then when the "child" query is updated, the "parent" tags are not extended to encompass the range of the child query. To resolve this, you would use [Startrange] to specify the same starting range for both the parent and the child query.

In the following example, the [Startrange] for both queries is placed to the left of the [AQ1] tag. This ensures that the complete tags for both queries will move in synch as the queries are populated with data. This is not a real implementation example; the only purpose is to show how the tags would be placed for the nested queries.



If this example was done *without* the [Startrange] tags, then the range for the child query would start at the [AQ2] tag, which means that AQ2 would populate downward but the [Stop] tag of AQ1 would not.

IMPORTANT: The [Startrange] tag cannot be placed "after" the [AQ#] tag. It must be placed "before" the [AQ#] tag, meaning that it must be either to the left of the [AQ#] tag (for vertical queries) or above the [AQ#] tag (for horizontal queries). The [Startrange] tag allows you to make the insert range larger by extending beyond the [AQ#] tag; it cannot be used to make the insert range smaller.

Additional Axiom query options

This section details some additional Axiom query options that are useful for certain specialized file configurations.

Batch processing for Axiom queries

By default, Axiom queries are processed in sequential order, one by one. Axiom walks through the sheets defined on the Control Sheet from left to right, and then processes the queries on each sheet in ascending order (AQ1 first, then AQ2, and so on).

You can optionally define batches of Axiom queries, so that the queries within the batch are processed concurrently instead of sequentially. You can assign two or more queries to the same batch if those queries are not dependent on each other and therefore can be run at the same time. Batching queries to be run in parallel can improve file performance, especially when the file has many queries, or has several process-intensive queries (such as queries that return a lot of data, or that use large calc methods).

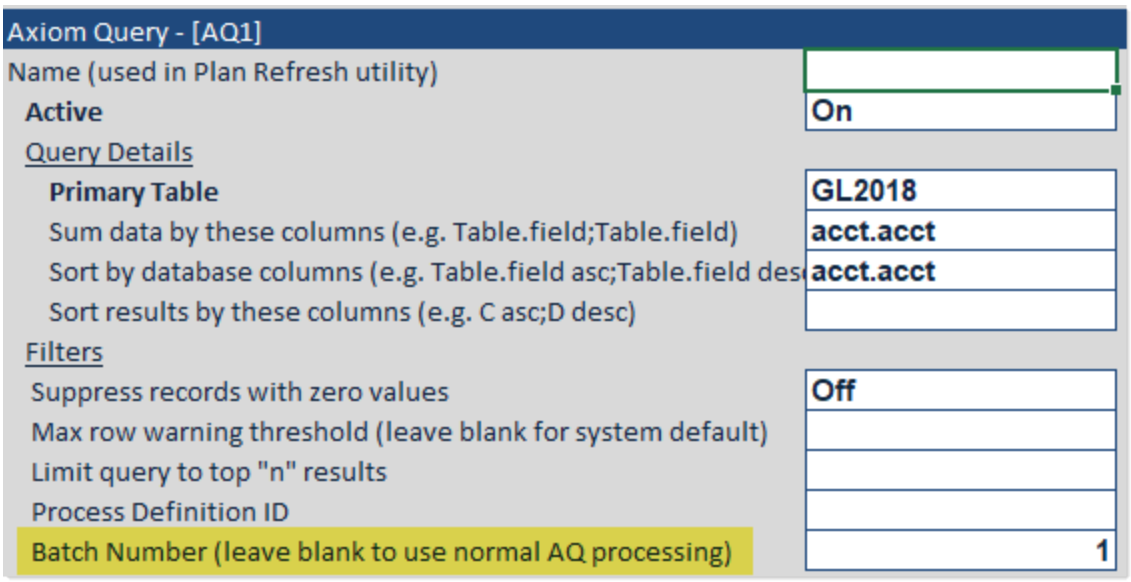
In order to process a query within a batch, you assign the query a batch number. Queries that have the same batch number are processed concurrently instead of sequentially. Batches are processed in ascending order, with the queries in batch 1 processed before the queries in batch 2, and so on.

Axiom query batches can be processed using two different approaches. The specific approach can be set on a per file basis.

- **Per-sheet batching**: By default, batches are processed per sheet. Each sheet is still processed in order, but the batched queries on the sheet are processed first, followed by the non-batched queries.
- **Cross-sheet batching**: If cross-sheet batching is enabled, then batches are processed across sheets. All queries that belong to a batch are processed concurrently in batch order, regardless of which sheet they are on. Once the batched queries are complete, non-batched queries are processed sheet-by-sheet as normal.

Enabling per-sheet batch processing

To enable per-sheet batch processing for Axiom queries, enter a batch number into the **Batch Number** field on the Control Sheet. This setting is defined per Axiom query, in the **Query Details** section.



Batch Number setting for an Axiom query

When batch numbers are defined for Axiom queries, query processing works as follows:

- Each sheet is still processed in order. For each sheet, queries with an assigned batch number are processed first. Batches are processed in ascending order (batch 1 first, then batch 2, and so on). All queries with the same batch number are processed in parallel. This means:
    - The database queries for all Axiom queries in the batch are made at the same time.
    - Axiom query data is placed in the sheet in the order that the database queries are completed. For example, if AQ1 and AQ2 are in the same batch, but the database query for AQ2 completes first, then data for AQ2 is placed in the sheet first.
- After all batches have been processed for a sheet, the remaining Axiom queries on that sheet are processed, in ascending order based on query number.

Per-Sheet Example

Imagine that the Axiom queries on a sheet are configured as follows:

Query	Batch
AQ1	1
AQ2	
AQ3	1
AQ4	2
AQ5	2
AQ6	

When this sheet is processed, the queries are run as follows:

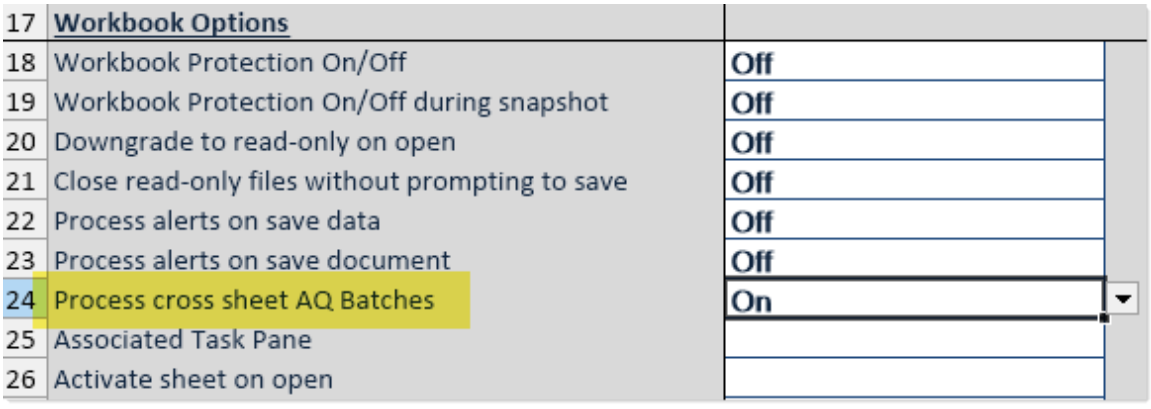
- AQ1 and AQ3 (batch 1) are processed first, in parallel.
- AQ4 and AQ5 (batch 2) are processed second, in parallel.
- The remaining queries are processed sequentially—AQ2 first, then AQ6.

You can re-use the same batch numbers on separate sheets, since each sheet is processed separately. For example, both the Lists sheet and the Report sheet can have a batch 1.

Enabling cross-sheet batch processing

To enable cross-sheet batch processing for Axiom queries:

- In the **Workbook Options** section of the Control Sheet, set **Process cross sheet AQ Batches** to **On**.



Setting to enable cross-sheet batching

- For all Axiom queries that you want to run in a batch, enter a batch number into the **Batch Number** field on the Control Sheet. This setting is defined per Axiom query, in the **Query Details** section.

When batch numbers are defined for Axiom queries and cross-sheet batch processing is enabled, query processing works as follows:

- Queries in batch 1 are processed first. This occurs regardless of which sheets the queries are on. Queries are processed in parallel, which means:
  - The database queries for all Axiom queries in the batch are made at the same time.
  - Axiom query data is placed in the sheet in the order that the database queries are completed. For example, if AQ1 and AQ2 are in the same batch, but the database query for AQ2 completes first, then data for AQ2 is placed in the sheet first.
- Queries in batch 2 are processed next, and so on, until all batches are completed.
- After all batches are completed, any remaining Axiom queries are processed using normal per sheet processing, in ascending order based on query number.

Cross-Sheet Example

Imagine that the Axiom queries in a file are configured as follows:

Sheet1		Sheet2	
Query	**Batch**	**Query**	**Batch**
AQ1	1	AQ1	2
AQ2	2	AQ2	1
AQ3	1	AQ3	2
AQ4	2	AQ4	
AQ5			

When this file is processed, the queries are run as follows:

- The queries in batch 1 are processed first, in parallel. This includes AQ1 and AQ3 from Sheet1, and AQ2 from Sheet2.
- The queries in batch 2 are processed next, in parallel. This includes AQ2 and AQ4 from Sheet1, and AQ1and AQ3 from Sheet2.
- Normal per-sheet query processing now begins. AQ5 on Sheet1 is processed first, then AQ4 on Sheet2.

Requirements and limitations

If a query is dependent on another query, then either the dependent query should not be assigned to a batch, or it should be assigned to a separate batch that runs after the "parent" query.

For example, imagine that AQ1 is a rebuild query that brings in a list of departments, and AQ2 is an update-only query that brings in data for those departments. In this case, AQ1 must be run before AQ2. If there are other queries that can be run in parallel with AQ1, then you can assign AQ1 and the other queries to batch 1, and leave the batch number for AQ2 blank. Or, if AQ2 can be run in parallel with other queries that are dependent on the batch 1 queries, then AQ2 can be assigned to batch 2. But if both AQ1 and AQ2 are assigned to batch 1, then the queries will not work as expected.

Batch processing is not supported for the following types of queries:

- Queries where the primary table is a system table, such as Axiom.Columns. These queries cannot be run in parallel.
- All queries run during file processing. File processing does not support running queries in parallel.

The following query features do not apply when using batch processing:

- **Zero AQs on Save** is ignored for any query that is assigned to a batch.
- **Refresh Control Sheet between every AQ** is ignored for batch processing. The Control Sheet is not refreshed and settings are not re-read during a batch, or in between batches.

Running an individual Axiom query block on demand

You can set up a sheet so that users can run individual Axiom query blocks on demand, using the RunAxiomQueryBlock function. Users can double-click the cell with the function to run the query and populate the block. By default, users can double-click the cell again to zero the query and delete the rows in the block.

The intent of the function is to allow an Axiom query block to be run as needed by the user, instead of pre-populating the worksheet with data that may not be needed. Deferring the query until it is needed can improve performance, in cases where the worksheet does not always require the data to be present.

When using the default "collapse" behavior, the function can be used to simulate expanding and collapsing a set of rows within the worksheet, where the rows are built out via Axiom query. Instead of bringing all rows into the worksheet and then selectively hiding or showing certain rows (which can be performance-intensive), you can dynamically populate and clear blocks of rows as needed. This approach can be used to dynamically show detail rows in a section, or to perform in-sheet drilling. Alternatively you can opt to suppress the collapse behavior, so that double-clicking the function on a populated block has no effect. This option is intended to support rows with user inputs, so that the inputs are not inadvertently deleted before the user has a chance to save.

An Axiom query "block" means a set of `[AQ]` and `[Stop]` tags (data range tags). Each `[AQ]` tag can have a filter that limits the rows for that particular block. For example, you could have multiple Axiom query blocks in a sheet, where each block is populated with rows relating to a particular department, account, entity, job code, or any other dimension or grouping that you want to show in the sheet. You can associate a RunAxiomQueryBlock function with each block, so that users can double-click the function to populate blocks as needed.

Setting up RunAxiomQueryBlock

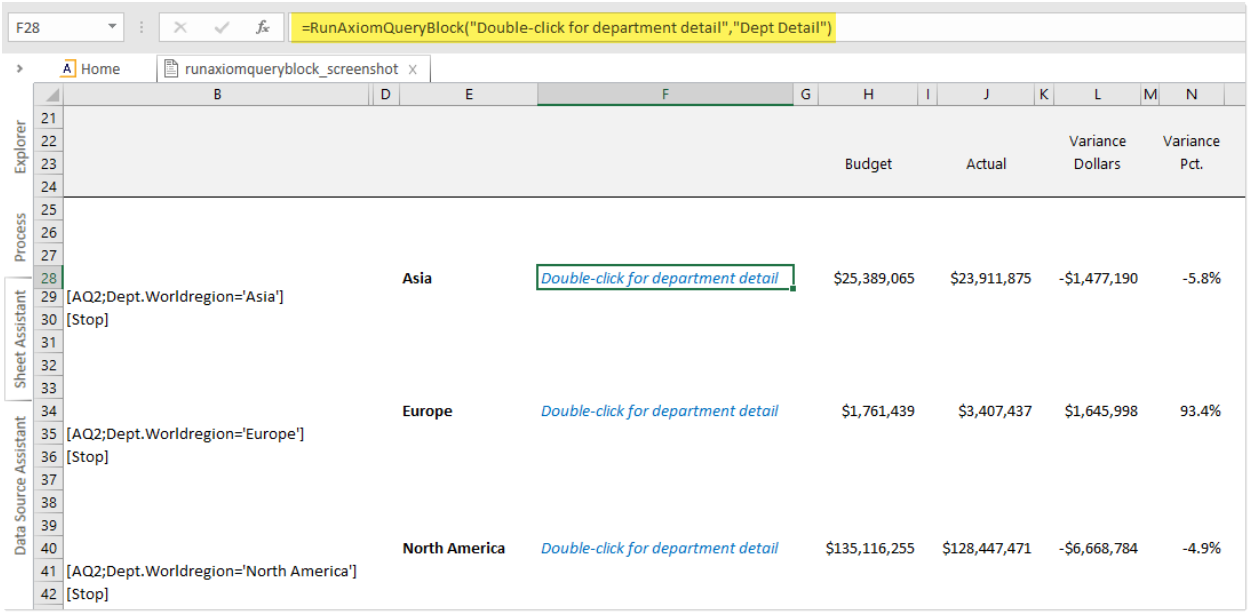
To enable running an Axiom query block by double-clicking a cell in the sheet, use the RunAxiomQueryBlock function. This function uses the following syntax:

```
RunAxiomQueryBlock("DisplayText", "AxiomQueryName", "NestedQueryNames",
SuppressCollapse)
```

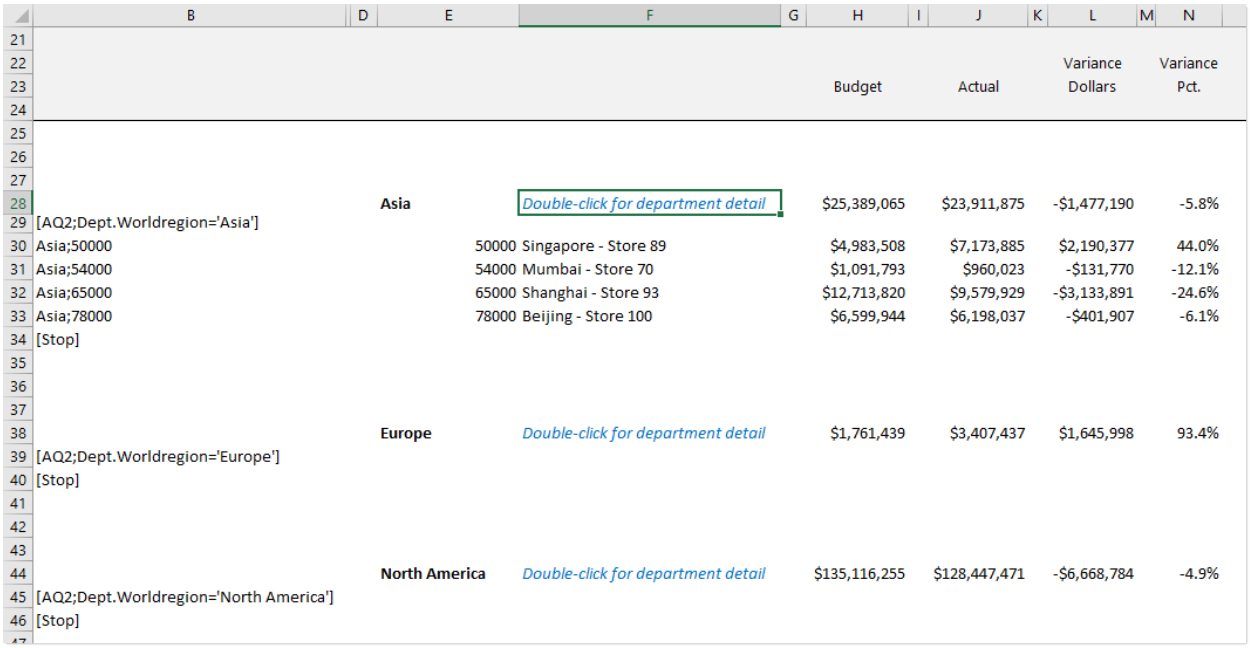
- **DisplayText** is the text to display in the cell.
- **AxiomQueryName** is the name of the Axiom query that the `[AQ]` block belongs to. For more information on query requirements and behavior, see Setting up Axiom queries for use with RunAxiomQueryBlock.
- **NestedQueryNames** is a list of one or more nested Axiom queries to run after the primary Axiom query is run. Separate multiple query names with semi-colons. Nested queries are run in the order listed.
- **SuppressCollapse** indicates whether users can zero populated blocks.
  - If FALSE (default), then double-clicking the cell when the target Axiom query block is already populated causes the query to be zeroed, meaning the rows are deleted. Once the rows are deleted, the user can double-click again to populate the rows, and so on. This is intended for use when the Axiom query block contains display-only data (no user inputs), so that the user can "expand" and "collapse" the data block as needed.
  - If TRUE, then double-clicking the cell when the target Axiom query block is already populated does nothing. This is intended for use when the rows in the Axiom query block allow user inputs to be saved. In this case, you do not want to delete the rows because the user may not have saved yet.

The RunAxiomQueryBlock function must be placed in the same row as the `[AQ]` tag for the block that you want to run, or above it. When a user double-clicks the cell with the RunAxiomQueryBlock function, Axiom looks in the insert control column for the specified Axiom query, and runs the first `[AQ]` block that it finds at the current row or lower.

In the following example, an `[AQ]` tag for the Axiom query named Dept Detail is located in row 29. The RunAxiomQueryBlock function has been placed in the row above it in order to populate this block. The function could also be placed on the same row as the tag.



When you double-click on the cell containing the function, the query is run and the target block is populated. Notice that the other `[AQ]` blocks were not populated, even though they belong to the same Axiom query.

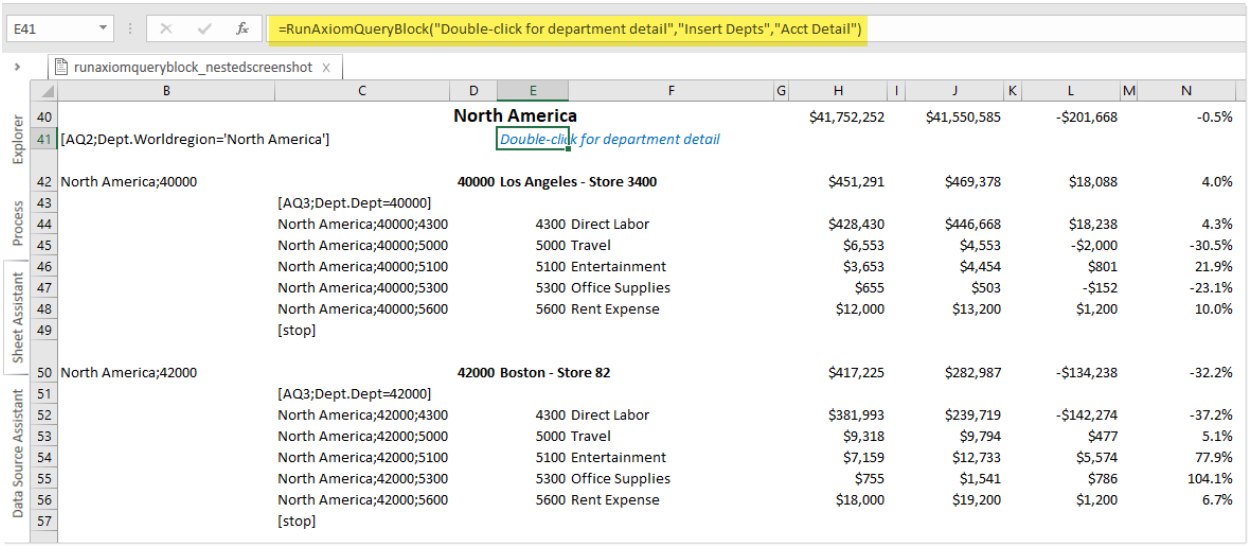


Because this function uses the default collapse behavior, you can double-click the cell again to zero the query and remove the rows. The sheet would then look exactly the same as it did in the first screenshot, with no rows in the Asia section. If instead the fourth parameter was set to False, then double-clicking the cell again would have no effect.

Running nested Axiom queries using RunAxiomQueryBlock

A nested Axiom query is when one "parent" query builds out the `[AQ]` tags of a "child" query. If the primary Axiom query for the RunAxiomQueryBlock function builds out the tags for one or more nested queries, you can also run those nested queries. Additionally, if a nested query builds out the tags for another nested query, that query can also be run.

In this example, the primary Axiom query (AQ2 - Insert Depts) brings in the departments, and then a nested Axiom query (AQ3 - Acct Detail) brings in accounts relating to each department. To run both queries, the nested query is listed in the third parameter of the function.



When the function is used, the block for the parent Axiom query is first populated. For each nested query listed (in the order listed), the following occurs:

- Axiom looks for `[AQ]` tags for the nested query within the rows of the primary block. The primary block starts at the `[AQ]` tag and ends at the `[Stop]` tag.
- If tags are found, the nested query block is run. If tags are not found, the process moves on to the next nested query in the list and does not error.
- Because each nested query may add rows to the original block, the size of the block is re-evaluated after each nested query is run. This means that one nested query can build out the tags for a subsequent nested query.

Setting up Axiom queries for use with RunAxiomQueryBlock

Axiom queries listed in the RunAxiomQueryBlock function must meet the following setup requirements:

- Queries must be defined on the same sheet as the function. The function cannot be used to run a query on another sheet.
- Queries must be standard vertical queries. Horizontal queries cannot be run using this function.
- The primary query must use the **Rebuild** refresh type. Nested queries can use any refresh type.

The queries are not required to be active, and do not require any particular refresh behavior settings. You can make the queries inactive if the only time they should be run is by using RunAxiomQueryBlock.

When the function is used on an empty `[AQ]` block, the specified Axiom queries are run (primary query plus any listed nested queries) and the block is populated. By default, double-clicking the function again causes the queries to be zeroed—meaning, the rows in the block are deleted, and the block is once again empty. In this case, the block can be toggled between "expanded" and "collapsed" as many times as needed. When the optional "suppress collapse" parameter is enabled, then no action occurs if the block is already populated.

IMPORTANT: If the rows in the Axiom query block contain user input cells, the default "collapse behavior" should not be used. If a user expands a block and enters data into input cells, then collapses the block without saving first, the user's inputs will be lost when the rows in the block are deleted.

When an Axiom query is run using RunAxiomQueryBlock, the database query is specially targeted so that it only brings back data for the block to be populated. Any filter on the `[AQ]` tag is added to the database query, to improve query performance and eliminate any unmatched data. This is different from normal Axiom query behavior, where data range filters are not added to the database query and only impact the placement of data in the sheet.

Note the following regarding Axiom query behavior when using RunAxiomQueryBlock:

- Views are reset after running Axiom queries using RunAxiomQueryBlock. This means that if some rows populated by the query are tagged to be hidden by the currently applied view, those rows will be hidden.
- The Axiom query order and any batch settings are ignored when running Axiom queries using RunAxiomQueryBlock. Nested Axiom queries are run in the order they are listed.
- The selective running of the Axiom query block is not considered to be a full refresh. Refresh-related features are not processed, such as refresh variables and data lookups.
- In plan files, users do not need to have the **Run Axiom Queries** permission in order to use RunAxiomQueryBlock. Queries run using this function are exempt from this security requirement.
- If you need to dynamically enable or disable this feature based on some condition, the RunAxiomQueryBlock function should be wrapped in an IF function. Disabling the Axiom queries will not disable the feature, since the queries are not required to be active.

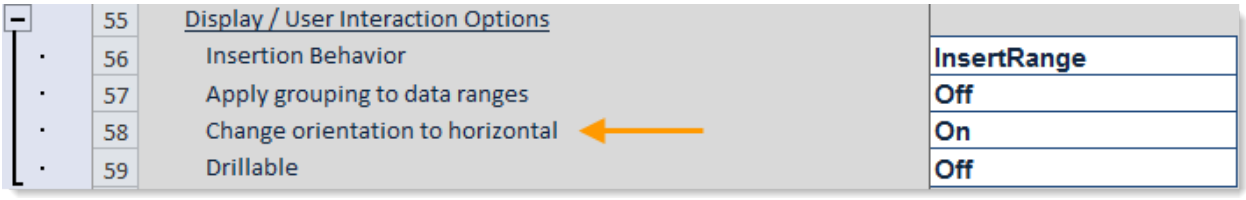
Using horizontal Axiom queries

Standard Axiom queries are oriented vertically, with data coming into the sheet as rows. If desired, you can configure an Axiom query so that it is oriented horizontally instead, with data coming into the sheet as columns.

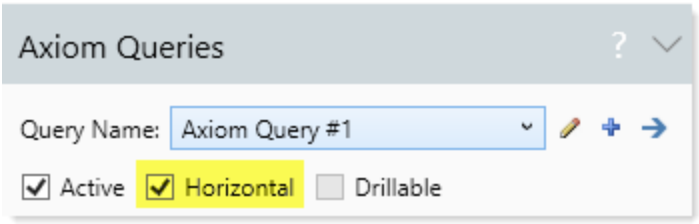
Horizontal Axiom queries can be used for any purpose, but one of the primary use cases is to use a horizontal query to build out components for a second, vertical Axiom query. For example, a horizontal query can be used to build out the field definition for a vertical query. Essentially, if the *columns* of the query need to be dynamic, a horizontal query should be used.

Configuring a query as horizontal

To configure an Axiom query as horizontal, set **Change orientation to horizontal** to **On**. In the Control Sheet, this setting is located in the **Display / User Interaction Options** section.



This setting is also available in the Sheet Assistant by selecting the **Horizontal** check box.



Once a query has been configured as horizontal, the orientation of the control rows and columns changes to match. For example, where standard vertical queries use field definition rows, horizontal queries use field definition columns. The Sheet Assistant updates dynamically for this changed orientation, so that you only see the applicable settings. When editing the Control Sheet directly, the applicable settings are located in the **Horizontal Configuration** section.

Any settings defined in the **Vertical Configuration** section are ignored.

Limitations of horizontal queries

Horizontal queries have the following limitations:

- Grouping does not apply to horizontal queries.
- Spreadsheet sorting does not apply to horizontal queries. There is no horizontal equivalent to the **Sort results by these columns** option; the setting is ignored.
- Data resulting only from a horizontal query cannot be drilled. However, the horizontal query is completely ignored as a drilling context, so if the data has another eligible context (from a vertical query, or by using GetData functions in the query) then that data can be drilled.
- Calc method libraries cannot be used with horizontal queries; only in-sheet calc methods can be used.

- If the horizontal query is in a template, plan file, or file group utility, then the insertion behavior must be Insert Range if the sheet has a corresponding calc method library. Inserting full columns is not supported in this context, as it would get the sheet out of sync with the calc method library.

All other Axiom query features apply to horizontal queries in the same way as vertical queries, within the context of the different orientation.

Design considerations for horizontal queries

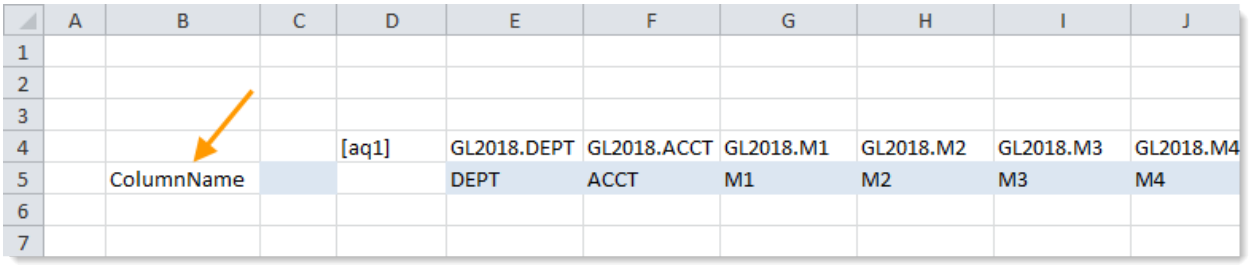
One of the most common use cases for horizontal queries is to build out dynamic columns to be used within a standard vertical query. For example, the horizontal query can build out the field definition for the vertical query.

NOTES:

- For each sheet, Axiom queries are processed in numerical order (AQ1 first, then AQ2, etc.). If you have a configuration where a vertical query is dependent on a horizontal query, the horizontal query must be processed first.
- If you are using a horizontal query in conjunction with a vertical query or other report content, then you most likely want to set the insertion behavior for the query to InsertRange. When the query is refreshed, data will be inserted by inserting new cells within the range, rather than inserting new columns (thereby leaving content above and below the query intact). You must use the additional `[Stoprange]` tag when using this option. For more information, see Insertion options for Axiom queries and Defining the insertion range when using Insert Range.

If you are building a horizontal query for the purposes of creating a field definition row for a standard vertical Axiom query, consider how you will use the horizontal query results for the second query.

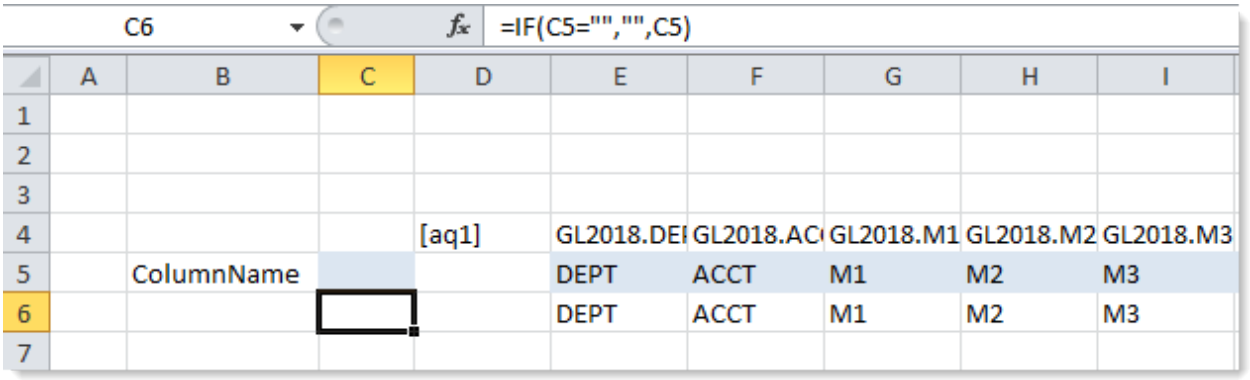
In the following example, you cannot specify row 5 as the field definition row for the vertical query, because the "ColumnName" field definition for the horizontal query will not be recognized as a valid entry for the second query.



	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4				[aq1]	GL2018.DEPT	GL2018.ACCT	GL2018.M1	GL2018.M2	GL2018.M3	GL2018.M4
5		ColumnName			DEPT	ACCT	M1	M2	M3	M4
6										
7										

NOTE: In this example, we are querying the system table Axiom.Columns to return all columns for the GL2018 table, so "ColumnName" is a valid field definition code.

Instead, you can add a formula to the in-sheet calc method for the horizontal query, which copies the results down to the next row. In the following example, column C is the in-sheet calc method, and the formula has been placed in cell C6. The formula copies the horizontal query results in row 5 down to row 6 (and is configured to return blank if no content exists in row 5). Row 6 can now be specified as the field definition row for the vertical query.



In the examples so far, the horizontal results can be used "as is" because they are column names. If you use the horizontal query to return something else, such as departments, then you need to use a formula to create a relevant field definition row for the vertical query.

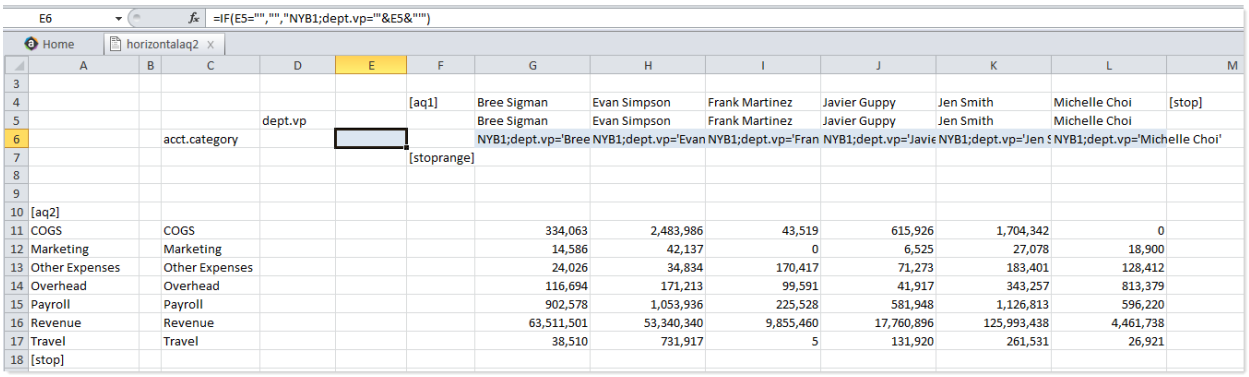
For example, imagine that you wanted to return the NYB1 data for each VP, with the VPs defining the columns (using column filters). If the horizontal query is returning VP names, then the formula that creates the field definition row for the second query would need to be something like the following:

```
=IF(E5="","","NYB1;dept.vp='"&E5&"'")
```

For columns where the horizontal query returns a VP name, this formula will create a field definition entry for the second query like so:

```
NYB1;dept.VP='Jones'
```

The following screenshot shows an example with a horizontal Axiom query populating the VP names for the field definition row of a second, vertical Axiom query. Row 6 is the field definition row for the second query.

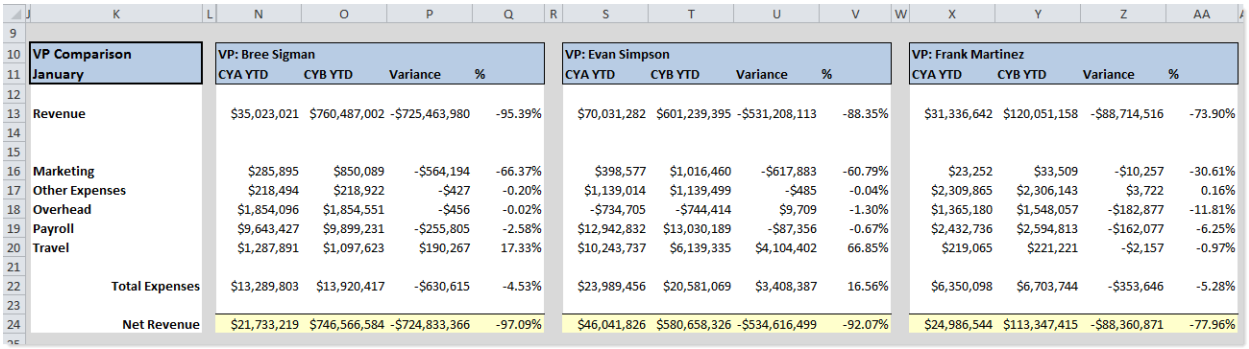


E6 =IF(E5="","","NYB1;dept.vp="&E5&"")

Home | horizontalaq2

	A	B	C	D	E	F	G	H	I	J	K	L	M
3													
4						[aq1]	Bree Sigman	Evan Simpson	Frank Martinez	Javier Guppy	Jen Smith	Michelle Choi	[stop]
5				dept.vp			Bree Sigman	Evan Simpson	Frank Martinez	Javier Guppy	Jen Smith	Michelle Choi	
6			acct.category				NYB1;dept.vp='Bree	NYB1;dept.vp='Evan	NYB1;dept.vp='Fran	NYB1;dept.vp='Javie	NYB1;dept.vp='Jen S	NYB1;dept.vp='Michelle Choi'	
7						[stoprange]							
8													
9													
10	[aq2]												
11	COGS		COGS				334,063	2,483,986	43,519	615,926	1,704,342	0	
12	Marketing		Marketing				14,586	42,137	0	6,525	27,078	18,900	
13	Other Expenses		Other Expenses				24,026	34,834	170,417	71,273	183,401	128,412	
14	Overhead		Overhead				116,694	171,213	99,591	41,917	343,257	813,379	
15	Payroll		Payroll				902,578	1,053,936	225,528	581,948	1,126,813	596,220	
16	Revenue		Revenue				63,511,501	53,340,340	9,855,460	17,760,896	125,993,438	4,461,738	
17	Travel		Travel				38,510	731,917	5	131,920	261,531	26,921	
18	[stop]												

You could take this further and use the horizontal query to set up multiple-column formatting:



	K	N	O	P	Q	S	T	U	V	X	Y	Z	AA
9													
10	**VP Comparison**	**VP: Bree Sigman**				**VP: Evan Simpson**				**VP: Frank Martinez**			
11	**January**	**CYA YTD**	**CYB YTD**	**Variance**	**%**	**CYA YTD**	**CYB YTD**	**Variance**	**%**	**CYA YTD**	**CYB YTD**	**Variance**	**%**
12													
13	**Revenue**	$35,023,021	$760,487,002	-$725,463,980	-95.39%	$70,031,282	$601,239,395	-$531,208,113	-88.35%	$31,336,642	$120,051,158	-$88,714,516	-73.90%
14													
15													
16	**Marketing**	$285,895	$850,089	-$564,194	-66.37%	$398,577	$1,016,460	-$617,883	-60.79%	$23,252	$33,509	-$10,257	-30.61%
17	**Other Expenses**	$218,494	$218,922	-$427	-0.20%	$1,139,014	$1,139,499	-$485	-0.04%	$2,309,865	$2,306,143	$3,722	0.16%
18	**Overhead**	$1,854,096	$1,854,551	-$456	-0.02%	-$734,705	-$744,414	$9,709	-1.30%	$1,365,180	$1,548,057	-$182,877	-11.81%
19	**Payroll**	$9,643,427	$9,899,231	-$255,805	-2.58%	$12,942,832	$13,030,189	-$87,356	-0.67%	$2,432,736	$2,594,813	-$162,077	-6.25%
20	**Travel**	$1,287,891	$1,097,623	$190,267	17.33%	$10,243,737	$6,139,335	$4,104,402	66.85%	$219,065	$221,221	-$2,157	-0.97%
21													
22	**Total Expenses**	$13,289,803	$13,920,417	-$630,615	-4.53%	$23,989,456	$20,581,069	$3,408,387	16.56%	$6,350,098	$6,703,744	-$353,646	-5.28%
23													
24	**Net Revenue**	$21,733,219	$746,566,584	-$724,833,366	-97.09%	$46,041,826	$580,658,326	-$534,616,499	-92.07%	$24,986,544	$113,347,415	-$88,360,871	-77.96%

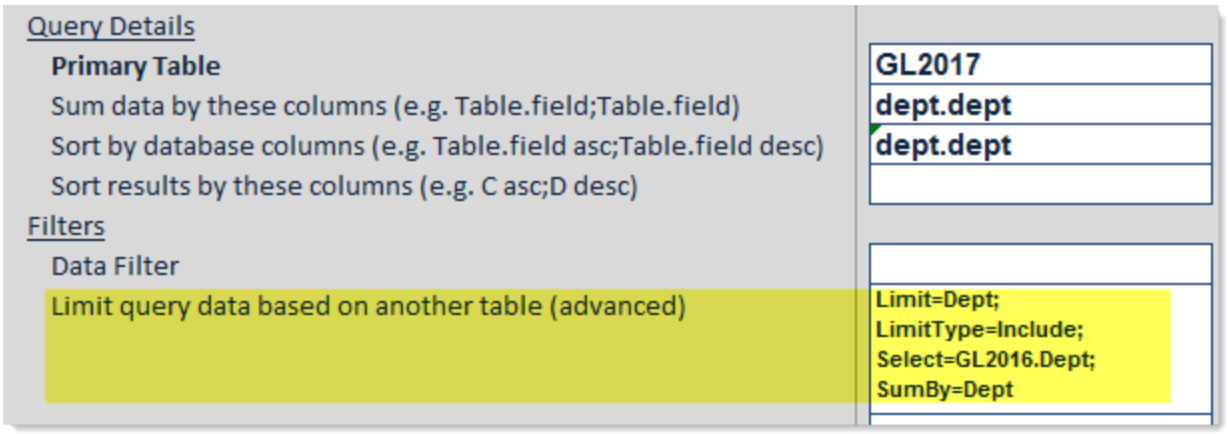
Limit the data in an Axiom query based on another query

You can limit the data in an Axiom query based upon another "pre-query". The limit query runs first to determine the overall set of data available to the Axiom query. The Axiom query then runs within the context of the available data.

NOTE: This is an advanced feature that should only be used by report designers who have a strong knowledge of your system's data structures. A good understanding of SQL query syntax is also helpful.

For example, you may want to create an Axiom query that only brings in items that are also used in another table. Or, you may want to only bring in items that are *not* used in another table, such as to create a pick list of unused items.

If you want to apply a limit query to an Axiom query, this is defined on the Control Sheet using the setting **Limit query data based on another table**.



Within this field, you enter special syntax to define the parameters of the limit query. In this example, the Axiom query is querying data from GL2017. We want to limit the data in the Axiom query to only those departments that are also used in the GL2016 table. The syntax used in the limit query setting identifies the following:

- The column to limit within the Axiom query
- The limitation behavior, to include or exclude matched records
- The table and column to "pre-query" to limit the data
- The "sum by" level for the limit query
- An optional filter to apply to the limit query

Limit query syntax

The parameters of the limit query are defined using the following syntax:

`Limit=`*`ColumninAQ`*`; LimitType=`*`Include/Exclude`*`; Select=`*`Table.Column`*`; Where=`*`Criteria`*`; Having=`*`Criteria`*`; Sumby=`*`Columns`*

Parameter	Description
Limit	The column to limit within the Axiom query. **NOTE:** The legacy syntax `LimitColumn` can also be used for this parameter instead of `Limit`. You may see this used in older files.
LimitType	Optional. The type of limitation you want to perform: • **Include**: Include all items that are matched by the limit query. This is the default limit behavior if this parameter is omitted. • **Exclude**: Exclude all items that are matched by the limit query. For example, if you want the Axiom query to include all departments that are also found in another table, you would use the Include limit behavior. If you want the Axiom query to exclude all departments that are also found in another table, you would use the Exclude behavior.

Parameter	Description
Select	The Table.Column that you want to compare the limit column against. This defines the primary table and column for the limit query. This column must contain the same type of data as the limit column so that the contents of both can be compared against each other. **NOTE:** The legacy syntax `Column` can also be used for this parameter instead of `Select`. You may see this used in older files.
Where	Optional. A criteria statement to filter the limit query data. You can use any filter criteria statement that would be valid against the table specified in the Select parameter. This filter has the same behavior as the Data Filter for Axiom queries. **NOTE:** The legacy syntax `Filter` can also be used for this parameter instead of `Where`. You may see this used in older files.
Having	Optional. A criteria statement to filter the limit query data *after* it has been grouped. For more information, see Using the Having parameter in a limit query.
Sumby	Optional. The "sum by" level (grouping level) for the limit query. If omitted, the same default sum by behavior applies as for Axiom queries. In most cases, the sum by level should match (or at least end with) the column specified in the Select parameter.

The order of the parameters does not matter. If optional parameters are omitted, you do *not* need to delimit the omitted parameters with "empty" semicolons.

NOTE: Calculated fields cannot be used in any of the limit query parameters.

Using multiple limit queries

If needed, you can use multiple limit query statements to limit the data in your Axiom query. This might be necessary if you want to limit the data in the query using the data from more than one table, or using more than one limit column.

When using more than one limit query statement, you must place each statement in curly brackets, and connect them using AND or OR. For example:

```
{Limit=EncounterID; Select=EncPhys.EncounterID;
Where=EncPhys.Physician.Name in ('Jones','Smith')} AND
{Limit=EncounterID; Select=EncStats.EncounterID; Where=EncStat.Value =
123}
```

In this example, the data in the main query will be limited to the encounter IDs in the EncPhys table that are associated with the listed physicians in the filter, and where those encounter IDs are also in the EncStats table associated with the code 123.

The curly brackets are only necessary when using multiple limit query statements, to designate the beginning and end of each statement. However, single statements with curly brackets are still valid.

Using the Having parameter in a limit query

Using the Having parameter, you can optionally filter the data in the limit query after it has been grouped. The Having parameter can be used alone or with the Where parameter to filter the data in the limit query.

Technically speaking, the difference between the two parameters is as follows:

- The Where parameter is applied as a WHERE clause to the resulting SQL query for the limit query. This is the same behavior as when using the Data Filter for an Axiom query. The WHERE clause is applied to determine the individual data rows to be included in the query, and then these rows are grouped according to the "sum by" level for the query (either as specified in the query parameters or assumed).
- The Having parameter is applied as a HAVING clause to the resulting SQL query for the limit query. It allows you to filter the data *after* it has been grouped, instead of before the grouping.

The Having parameter uses the following syntax:

AggregationType(*ColumnName*)=*Value*

Where *AggregationType* is the desired type of aggregation, such as sum or count. Any valid aggregation type and operator type supported for the HAVING clause in the SQL database query language can be used here.

For example, imagine the sum by for the query is by region, and you want to only include regions that include 10 or more stores. You could use a limit query statement with a Having parameter, such as:

```
Limit=Region; Select=Dept.Region; Where=Dept.Type='Store'; Having=Count
(Dept.Dept)>=10; SumBy=Dept.Region
```

Creating a limit query statement

If you want users to be able to create limit query statements and apply them to an Axiom query, there are several ways you can do this. The following features all support showing a wizard that allows users to create a limit query statement:

- AdvancedFilter refresh variable (for use in Axiom forms or spreadsheet Axiom files)
- ShowFilterWizardDialog function (for use in spreadsheet Axiom files)

- Filter Wizard command (for use in Axiom forms)

When the user's selections in the wizard are complete, the wizard generates a limit query statement using the syntax described above. You can then reference this statement in the **Limit query data based on another table** option. The limit query statement includes the following:

- The Limit column specified in the feature configuration. This determines the table columns available for selection in the wizard.
- The Select column selected by the user in the wizard.
- The Where filter created by the user in the wizard.

The wizard does not support user selections for any of the other parameters in the limit query statement. Since the LimitType is omitted, it is assumed to be Include. If the user is familiar with limit query syntax, they can manually edit the limit query statement before applying it to the file.

Organizing Axiom query data into ranked segments

You can organize your Axiom query data into ranked segments, so that you can easily see which records fall into the top, middle, and bottom of the data distribution.

For example, you may want to construct an Axiom query that returns the revenue for each store in Q1. Then you want to rank each store based on where they fall in five segments, where segment 1 contains the top performers and segment 5 contains the bottom performers.

Defining the segment parameters in the field definition

To segment your Axiom query data, you use special syntax in the Axiom query field definition. This syntax defines the number of segments and the database column to use for segmentation, and creates a special "virtual" column to report on the segment results.

The segment syntax is as follows:

`[Column=`*`VirtualSegmentColumn`*`; Type=Segment; Segments=`*`Number`*`; SegmentBy=`*`ColumnName`*`]`

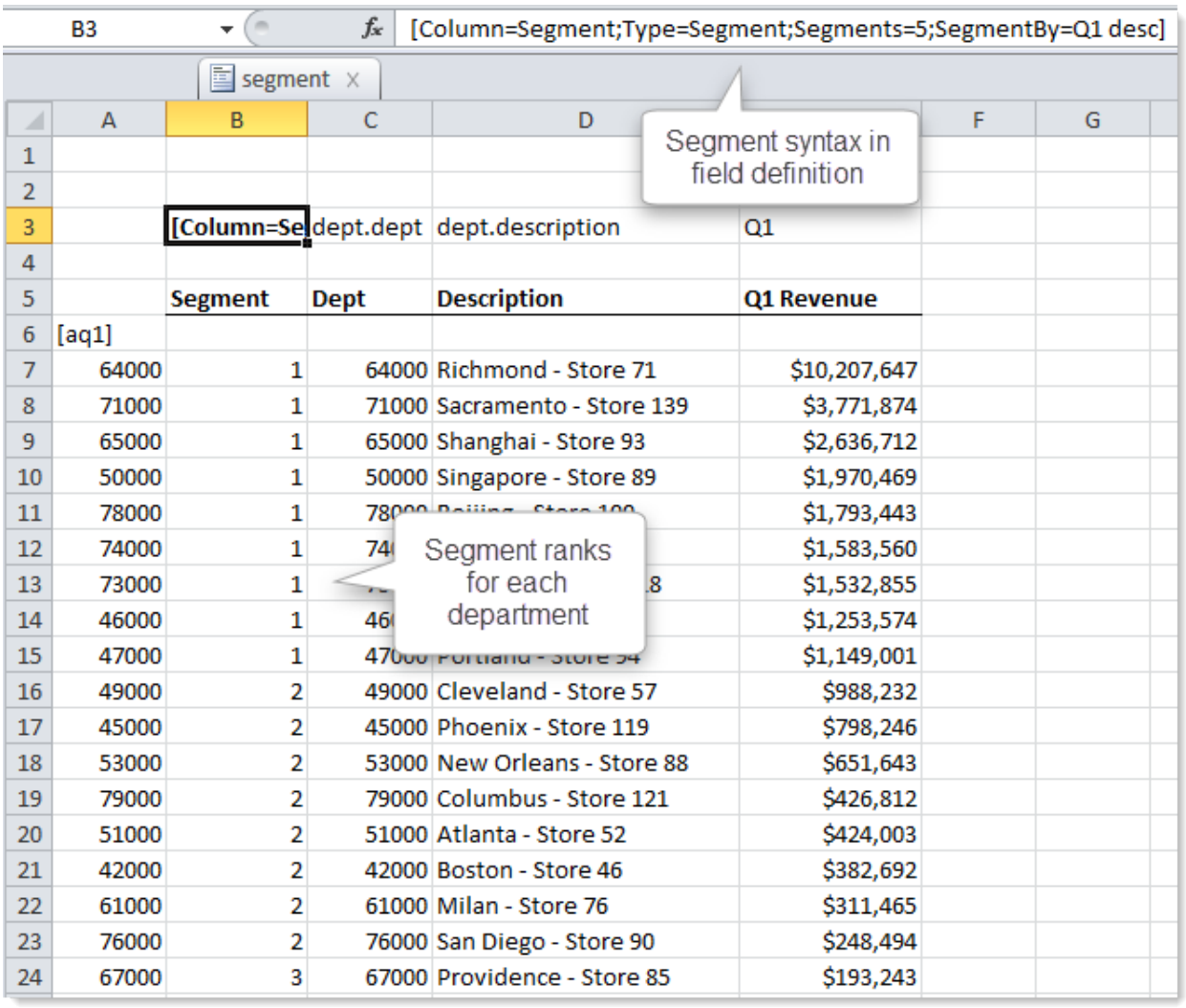
Parameter	Description
Column	The name of the virtual column to report the segment results. You can name this column anything you like—for example, you could name it Segment (for any number of segments) or Decile (if you are using 10 segments). When the query is run, this column will return the number of the segment that each data record in the query belongs to.
Type	Use the keyword `Segment` for this parameter. This parameter specifies the type of action to perform on this column, which is to segment the data.

Parameter	Description
Segments	The number of segments to organize the data into. Enter any whole number greater than 1. For example, if you want the data divided into 5 segments or 10 segments, enter 5 or 10.
SegmentBy	The database column(s) to be used as the basis of the segmentation. At the database level, this will sort the query data using the SegmentBy column(s), and then the segment rankings will be assigned based on this order. For example, if you want the records to be segmented based on their results in Q1, enter the column name Q1. The column name can optionally be followed by `Asc` or `Desc` to specify the sort order of the column values (this will impact whether low values are considered "top" segment or "bottom" segment). If you want to use multiple columns, separate the column names using commas. You can use column names and alias names in this setting. If you use a column name, the primary table is assumed if the table name is omitted. Values in the SegmentBy column(s) are evaluated using ascending order by default, which means that records with the lowest values will be in the "top" segment. If instead you want the records with the highest values to be in the top segment (such as in the example of segmenting based on revenue), then you should specify descending order.

In the previous example of segmenting stores by Q1 data, the field definition syntax would be as follows:

```
[Column=Segment; Type=Segment; Segments=5; SegmentBy=Q1 Desc]
```

The "sum by" for the Axiom query would be set as appropriate to return stores as the rows. For example, the sum by might be set to Dept.Dept with a data filter to restrict the results to departments that are stores. The data filter for the query would also be used to restrict the results to revenue.



	A	B	C	D	F	G
1						
2						
3		[Column=Se	dept.dept	dept.description	Q1	
4						
5		**Segment**	**Dept**	**Description**	**Q1 Revenue**	
6	[aq1]					
7	64000	1	64000	Richmond - Store 71	$10,207,647	
8	71000	1	71000	Sacramento - Store 139	$3,771,874	
9	65000	1	65000	Shanghai - Store 93	$2,636,712	
10	50000	1	50000	Singapore - Store 89	$1,970,469	
11	78000	1	78		$1,793,443	
12	74000	1	74		$1,583,560	
13	73000	1		.8	$1,532,855	
14	46000	1	46		$1,253,574	
15	47000	1	47		$1,149,001	
16	49000	2	49000	Cleveland - Store 57	$988,232	
17	45000	2	45000	Phoenix - Store 119	$798,246	
18	53000	2	53000	New Orleans - Store 88	$651,643	
19	79000	2	79000	Columbus - Store 121	$426,812	
20	51000	2	51000	Atlanta - Store 52	$424,003	
21	42000	2	42000	Boston - Store 46	$382,692	
22	61000	2	61000	Milan - Store 76	$311,465	
23	76000	2	76000	San Diego - Store 90	$248,494	
24	67000	3	67000	Providence - Store 85	$193,243	

Limitations and design considerations

Because the virtual segment column is not a real database column, it cannot be used in most "regular" Axiom query settings such as the sum by or the data filter. There is one exception—the column can be used for the database sort.

- **Sorting by segment**: If you want to sort the Axiom query results based on the segment numbers, you can use either the database sort or the spreadsheet sort. For the database sort, use the virtual column name as defined in the field definition syntax (for example, `Segment`).
- **Grouping by segment**: If you want to group the Axiom query results based on the segment (i.e. each row shows the sum of a particular segment), then you must use the **Post-Query Sum By** option in the Query Details section of the Control Sheet. Enter the virtual column name as defined in the field definition syntax (for example, `Segment`). The regular "sum by" setting does not recognize the virtual column and cannot use it to group results.

- **Filtering by segment**: If you want to filter the Axiom query results by segment—for example, only show departments that fall into segment 5—then you can use the **Post-Query Filter** option in the Query Details section of the Control Sheet. When defining the filter, use the virtual column name as defined in the field definition syntax, such as: `Segment=5`. Alternatively, you can filter by segment using data range filters—for example, to have one data range per segment, such as `[aq1; segment=5]`. The Axiom query data filter and the sheet filters do not recognize the virtual column and cannot use it to filter results.
- **Drilling**: Segmenting is not supported for use with drill-down drilling. For example, if you are using the post-query "sum by" to group by segment, you cannot drill down to the dimension elements in that segment. If you need to drill segmented data, you may be able to configure a custom drill to meet your needs.

Return the top N records for an Axiom query

You can configure an Axiom query so that it returns only the top N records for the query—such as top 5, top 10, or top 50. For example, you may want to see the top 10 departments based on revenue in Q1.

In order to return the top N records based on a particular criteria, you must:

- Enter the number of records that you want returned into the field **Limit query to top "n" results**. This setting is only located on the Control Sheet, in the **Query Details** section for the Axiom query.
- Set **Sort by database columns** to the column(s) that you want to use to determine the top results. This setting can be defined using either the Control Sheet or the Sheet Assistant; in the Sheet Assistant the setting is called **Data Sort**.)

When the Axiom query is run, it will return the first N records (in this example, 10) in the query. This means that the sort order is very important to determine which records you get. Remember that by default the sort is in ascending order. So if you specify CYA_Q1 as the sort column but do not specify an order, you will get the 10 smallest records in that column, not the 10 largest records. If you want the 10 largest records, you must specify "desc" for descending order.

If you want the results in the sheet to display sorted in a different order than the order used to specify the top N results, then you can also specify a spreadsheet sort (**Sort results by these columns**). This will sort the data after it has been inserted into the sheet, by the specified spreadsheet column(s).

NOTE: If the top N setting is used to limit the query, the **Max row warning threshold** is ignored.

▶ Top N example

The following screenshot shows how the query might be configured to return the top 10 departments based on revenue in Q1:

Query Details	
Primary Table	**GL2013**
Sum data by these columns (e.g. Table.field;Table.field)	**dept.dept**
Sort by database columns (e.g. Table.field asc;Table.field desc)	**CYA_Q1 desc**
Sort results by these columns (e.g. C asc;D desc)	
Data Filter	**acct.category='revenue'**
Limit query data based on another table (advanced)	
Suppress records with zero values	**Off**
Max row warning threshold (leave blank for system default)	
Limit query to top "n" results	**10**

- The "sum by" level is set to `dept.dept` to return department-level data.
- The database sort is set to sort the data by `CYA_Q1` in descending order.
- The data filter is set to limit the data to `acct.category='revenue'`.
- The top N limit is set to return the top `10` records for the query.

The following screenshot shows how the results of the Axiom query might look in the sheet:

	C	D	E	F
5		**Top 10 Departments by Revenue**		
6		**Q1 2013**		
7				
8		**Dept**	**Description**	**Q1 Revenue**
9				
10		64000	Richmond - Store 71	$4,217,837
11		71000	Sacramento - Store 139	$3,497,516
12		47000	Portland - Store 94	$2,764,724
13		65000	Shanghai - Store 93	$1,921,833
14		50000	Singapore - Store 89	$1,302,582
15		62000	Detroit - Store 102	$1,293,922
16		49000	Cleveland - Store 57	$1,217,878
17		73000	Nashville - Store 118	$1,154,063
18		46000	Chicago - Store 45	$958,492
19		40000	Los Angeles - Store 34	$910,936
20				
21			**Total:**	$19,239,785
22				

In this example, the results are sorted by the database sort level, in CYA_Q1 descending order. If you wanted the results to display in the sheet sorted by department number, then you could set **Sort results by these columns** to column D.

Converting data in an Axiom query

You can apply data conversion to an Axiom query, so that data returned from the query is converted according to the specified conversion target. For example, if a particular table stores data in U.S. dollars, you can convert the data as part of the query and return the values in Canadian dollars.

The table that you are querying must already be configured to support data conversions. For more information on setting up data conversions for a table, see the *System Administration Guide*.

▶ Setting up an Axiom query for data conversions

To convert data in an Axiom query, complete the following settings for the query on the Control Sheet:

Field	Description
Conversion Target	Enter the target for the conversion. For example, if the "source" data in the table is U.S. dollars and you want to convert the data to Canadian dollars, you would enter CAD as the conversion target. You can enter a fixed value that will apply to all records, or you can look up the value on a record by record basis from a grouping column in a lookup reference table. The fully qualified name must be used—for example, `DEPT.TargetCur`. **NOTE:** If you use a grouping column, it is recommended to also include the grouping column in the field definition for the query, to identify the conversion that was applied for each individual record. In either case, the conversion target must be a value that is found in the "to" column for the associated conversion table. The conversion table is determined on a column by column basis (for each data column entry in the field definition), using the conversion configuration for the associated table.
Conversion Scenario Override	If desired, specify a conversion scenario to override the default scenario for the data conversion. The entry must be a value that is found in the "scenario" column for the conversion table.
Conversion Type Override	If desired, specify a conversion type to override the default type for the data conversion. The entry must be a value that is found in the "type" column for the conversion table.

If an Axiom query is configured for data conversions, then at least one of the data tables in the query must be enabled for data conversions, or else the query is invalid and an error will result. If multiple tables are being queried, and some are enabled for conversions and some are not, the conversion settings will only apply to the tables that are enabled for conversions. Tables that are not enabled for conversions will return their data as is.

If the conversion target or either of the overrides are invalid, then zero values are returned. Data conversions will also return zero values if the conversion configuration for the table has invalid values, or if the conversion table itself has no valid rate entry for the relevant conversion and period (for example, blank entries in a rate column).

Multiple data tables and data conversions

Special considerations apply if you want to use data conversions with an Axiom query that queries multiple data tables.

Any data table that is configured for data conversions will be converted when the query is run. Axiom evaluates the data conversion on a column by column basis (for each data column in the field definition) and applies the unique conversion configuration for each associated table. So each table in the query can be associated with different conversion tables and can use different conversion settings, such as the default conversion type and scenario. However, every data table to be converted must use the same designated conversion target.

Example

Imagine that you want to query column CYA1 of the GL2020 table and column LYA1 of the GL2019 table, and you want to convert all of the data from USD to CAD. Because the two tables cover different years of data, they likely use different conversion tables to allow for the different currency conversion rates for each year. Axiom will honor the conversion configuration for each individual table, and will apply the appropriate rates to the conversion. However, both conversion tables must have a "to" entry of CAD, because each Axiom query can have only one conversion target that applies to the entire query.

If one of the tables did not contain the "to" entry of CAD, the columns for that table would return zero values in the Axiom query. Whenever a conversion setting is invalid (such as pointing to a "to" entry that does not exist in the conversion table), the result is zero values.

Similarly, if you specify a scenario override or a type override, the specified override value must exist in all of the associated conversion tables for each data table in the query.

You can set up an Axiom query that includes converted tables and non-converted tables. If a table is not enabled for data conversions, then the conversion settings are ignored and data is returned "as is" for those tables. However, if a table is enabled for data conversions, then it will always be converted if the Axiom query conversion settings are used. You cannot selectively apply conversions on a table by table or column by column basis within a single Axiom query.

If you want to query converted and non-converted data from tables that are enabled for conversions (for example, to show USD and CAD values side-by-side, for the same period), then you can do one of the following:

- Use GetData functions instead of an Axiom query. Each individual GetData function can be flagged to convert or not to convert.

- Use two parallel Axiom queries, with one query configured to convert and one not to convert. For example, the first query would be set to "rebuild" and would populate the report with the desired data records and the unconverted values. The second query would be set to "update" and would update each record to add the converted values.

Using spreadsheet grouping with an Axiom query

Grouping is a Microsoft Excel feature that allows you to group multiple rows together. Once rows are grouped, you can expand and collapse the grouped sections by using the plus and minus icons displayed along the side of the spreadsheet. This feature is also known as "outlining."

Using grouping, you can present a lot of information in a small amount of space. Typically the "anchor" rows of each group present some kind of subtotals or header information, and then you can expand the group to see the detailed data.

You can enable grouping for an Axiom query, and then the results of each *data range* will be automatically grouped. This feature is best suited for queries that use multiple data ranges (with a different filter defined for each data range), or reports that use multiple nested Axiom queries.

NOTES:

- Grouping is only supported for use with standard vertical queries. This setting is ignored for horizontal queries.
- Only six Axiom queries can be enabled for grouping per sheet.

To enable grouping for an Axiom query:

- In the Sheet Assistant, in the Refresh Settings section for the query, select **Apply Groupings**.

This setting is located on the Control Sheet in the Display/User Interaction Options section for the query, as **Apply grouping to data ranges**.

When you refresh the query, each data range in that query is automatically grouped. The anchor row for the grouping is the row that contains the `[AQ#]` tag. All rows in the data range are grouped, including the row containing the `[Stop]` tag.

In the spreadsheet, a plus sign displays to the left of the anchor row. You can click on the plus sign to expand the section and see all the detail rows. If a section is expanded, you can click on the minus sign to collapse it. You can also use the "outline level" icons at the top of the grouping sidebar.

Note the following query behavior if grouping is enabled:

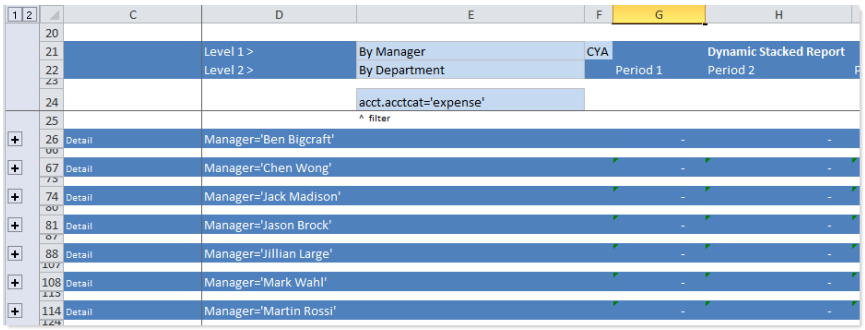
- Grouping is applied if the refresh option for the query is set to rebuild or insert. Any expanded sections are collapsed when the query is refreshed.
- If the query only updates existing data, grouping is not applied. However, any previously existing groupings will not be removed.

NOTE: If the sheet also uses a sheet view, note that applying a sheet view causes groupings to become expanded. You can work around this by tagging the rows within the grouping to be hidden by the view. You should hide all rows in the grouping except the row with the plus sign.

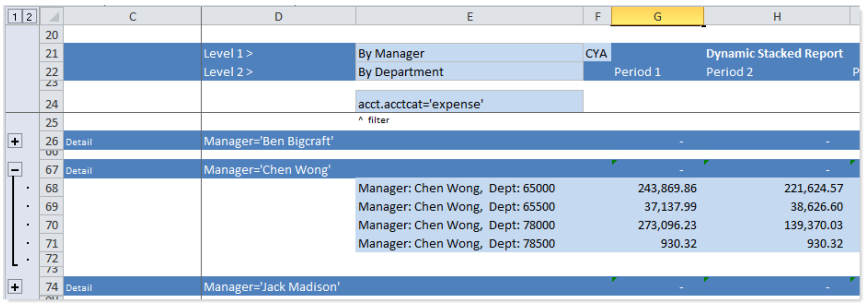
Grouping example

This example shows grouping being applied to multiple data ranges. Each data range in this query has a filter that limits the data to a particular manager.

The grouping controls display in a sidebar to the left of the spreadsheet:



To expand a section, click on the plus icon in the sidebar:



You can also use the "outline level" icons at the top of the sidebar. For example, clicking outline level 2 would expand all of the sections. To collapse all of the sections, click outline level 1.

Using an Axiom query to return system information

You can use an Axiom query to return system information within an Axiom file, such as:

- Table metadata (tables, columns, calculated fields, aliases, hierarchies, table types)
- Security information (users, roles, permissions)
- File group information (file groups, variables, plan file attachments, aliases)
- Process information (process events, process tasks, process steps)
- Other system information (configuration settings, alert notifications, file access, etc.)

This information is held in various system tables. A system table is a table maintained by Axiom, as opposed to the client-defined tables defined in the Table Library. System tables are named using the convention Axiom.*TableName*—for example, Axiom.Aliases is the system table that holds information on column aliases.

For the full list of queryable tables and column names, see **Axiom Help: Reference > System tables**.

Requirements and limitations

To query system information using an Axiom query, you use the appropriate Axiom.*TableName* as the **Primary Table**, and then use columns from that table in the field definition. For example, to query alias properties, the primary table is Axiom.Aliases, and the "column name" to return alias names is Alias.

NOTE: Column names from system tables should never be fully qualified. For example, always use just the column name of `Alias`, not `Axiom.Aliases.Alias`.

The Axiom query is otherwise set up as normal. Note the following considerations and limitations:

- **Invalid settings**: The following Axiom query settings do not apply when querying system information:
  - **Sum by**: The "sum by" level should be left blank when the primary table is a system table, because the sum level is inherent to the table and cannot be changed. There is one exception—you can change the sum by when querying the Axiom.ProductColumns table.
  - **Column filters**: Column filters cannot be used on field definition entries when the primary table is a system table.
  - **Batch number**: Batch processing cannot be used on a query when the primary table is a system table.

  Other advanced settings are not supported with system tables, such as limit queries or "top n".
- **Filtering results**: You can use the Axiom query's main **Data Filter** and/or data range filters to filter the data returned by the query. To define a filter, use the column names for the information category that you are currently querying. For example, if you are querying **Axiom.Aliases** and you want to see only the aliases defined for table GL2020, you could use the following filter in either location: `TableName='GL2020'`

  NOTE: If the column that you want to filter returns True/False, then in the filter criteria statement you can use either the text values True and False, or the numeric equivalent (1=True, 0=False). For example, `IsAdmin='True'` and `IsAdmin=1` will both work when querying **Axiom.Principals**.
- **Preview Axiom query data**: When previewing Axiom query data for a system table, depending on the table being queried the preview may show all columns in the table, or only the columns in the field definition.

- **Single table only:** The Axiom query can only use fields from the system table designated as the primary table. It is not possible to join two system tables, or to join a system table with a client-defined table.

Querying a hyperlink column

If hyperlink data is stored in a designated hyperlink column, an Axiom query can return this data using clickable hyperlinks.

For example, you might have a Parts table where each part is associated with a hyperlink to a specifications sheet. Or you might have a table that holds detailed transactional data, where each record is associated with a hyperlink to an invoice. When an Axiom query includes this Specification column or Invoice column, the contents of the column are automatically converted to clickable hyperlinks (rather than requiring you to manually convert the document data into a hyperlink).

For information on how to set up a hyperlink column in a table, see the *System Administration Guide*.

Using the hyperlink column in an Axiom query

When a hyperlink column is included in an Axiom query field definition, its values are returned as follows:

- If the column value starts with http, https, or file, the value is automatically wrapped in a Hyperlink function. For example:

  ```
  =HYPERLINK("http://partserver/part1spec.pdf", "Part Specification")
  ```

  Where `http://partserver/part1spec.pdf` is the value stored in the column and `Part Specification` is the text defined as the Hyperlink Label for the column. The equivalent file example would look like the following:

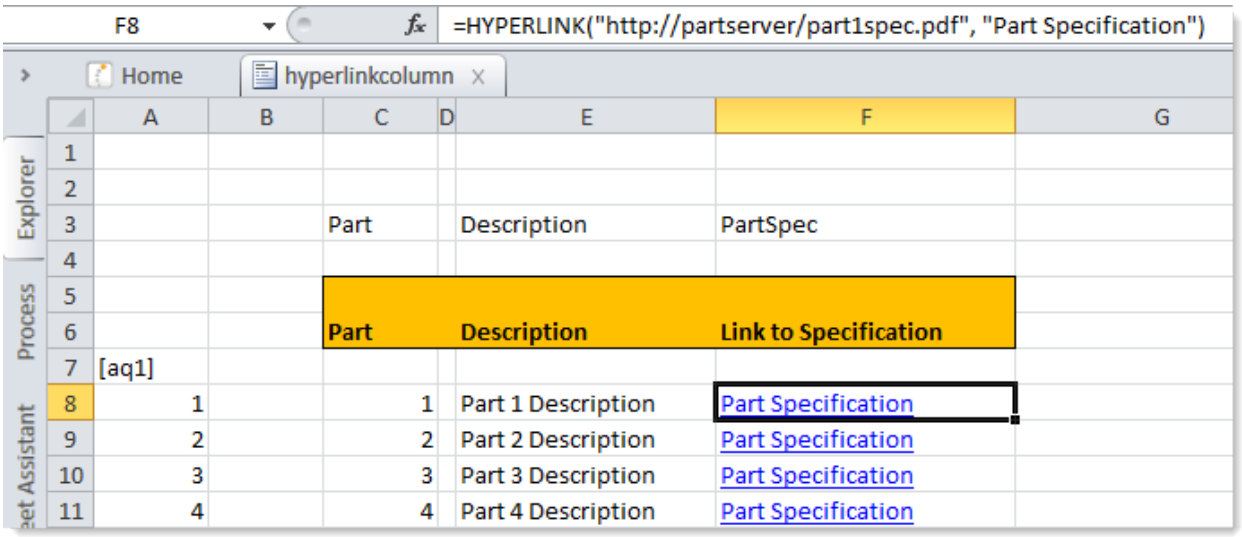
  ```
  =HYPERLINK("file://\\partserver\partspecs\part1spec.pdf", "Part Specification")
  ```

- Otherwise, the column value is automatically wrapped in a GetDocument function. Axiom assumes that any value that does not fall into the first category is a valid Axiom file path. For example:

  ```
  =GetDocument("Part Specification", "\Axiom\Reports Library\PartSpecs\part18spec.pdf")
  ```

  Where `Part Specification` is the text defined as the Hyperlink Label for the column, and `\Axiom\Reports Library\PartSpecs\part18spec.pdf` is the value stored in the column.

The following screenshot shows an example Axiom query. The PartSpec column is a designated hyperlink column. When the query is run, the column value is automatically converted to a hyperlink as shown in the formula bar. It is not necessary to manually convert the data using the in-sheet calc method.



Note the following design considerations:

- Microsoft Excel automatically formats hyperlink cells as blue font with underline text. The Windows Client does not. If a hyperlink is generated and saved in one client, then that hyperlink will continue to display as it was saved regardless of which client it is subsequently viewed in. Because of this, you may want to manually format the cell using the in-sheet calc method so that the hyperlink always displays in the same format.
- No formatting is automatically applied to the GetDocument functions. If you want these functions to display as if they were hyperlinks (blue font with underline text) then you must manually format the cell using the in-sheet calc method.
- Hyperlinks are launched with a single click, but GetDocument functions require double-clicking. If you are using GetDocument functions, you may want to indicate this behavior in the Hyperlink Label for the column (i.e. "Double-click to open specification"), or as user instruction within the file.
- Hyperlink functions using the file protocol are not supported for use in the Windows Client. These functions can only be used in the Excel Client.

Alternate query syntax to return the raw value

You can optionally override the default query behavior and instead return the raw value stored in the hyperlink column. You might do this if:

- You are creating a Save Type 1 report to populate and maintain the values in the hyperlink column. In this case, you must be sure to return the raw value from the column and save back a raw value.
- You are creating a formatted grid for use in an Axiom form. In this case you must manually return the raw value so that you can reference it in an HREF tag. See the following section for more information.

To return the raw value using an Axiom query, enter the following syntax in the field definition:

```
[Column=ColumnName;HyperlinkFormat=Raw]
```

For example, if the column is named PartSpec then you would enter the following:

```
[Column=Part.PartSpec;HyperlinkFormat=Raw]
```

You can also enter the column name as just PartSpec if the primary table of the Axiom query is the Part table, as shown in the following example:

	A	B	C	D	E	F	G
1							
2							
3			Part		Description	PartSpec	[Column=PartSpec;HyperlinkFormat=Raw]
4							
5							
6			**Part**		**Description**	**Link to Specification**	**Raw Value**
7	[aq1]						
8	1		1		Part 1 Description	Part Specification	http://partserver/part1spec.pdf
9	2		2		Part 2 Description	Part Specification	http://partserver/part2spec.pdf
10	3		3		Part 3 Description	Part Specification	http://partserver/part3spec.pdf
11	4		4		Part 4 Description	Part Specification	http://partserver/part4spec.pdf

NOTE: You can also use GetData to query a hyperlink column. In this case the raw value is always returned. You can wrap this result in a Hyperlink function or a GetDocument function to provide a clickable hyperlink.

Using hyperlink column values in Axiom forms

The Hyperlink and GetDocument functions generated by the Axiom query can only be used within the Excel Client and the Windows Client. Axiom forms do not render these functions as clickable hyperlinks.

If the hyperlink column contents are web URLs, then you can still use this data in Axiom forms. To do this, you must return the raw value instead of the converted value as described in the previous section. You can then reference the raw value in an HREF tag (or in a Hyperlink component) to display a clickable hyperlink for use in the form.

However, there is no equivalent option if the hyperlink column holds Axiom file paths. GetDocument functions are not usable in Axiom forms, and there is no way to convert the file path to a URL. A document ID is required to generate a URL using GetDocument hyperlink. You can generate hyperlinks for these documents in different ways (such as by querying Axiom.FileSystemInfo), but not by querying the hyperlink column.

Axiom query design considerations

This topic details some general design considerations for Axiom queries.

Processing order of Axiom queries

Axiom queries are processed in the following order:

- When refreshing the entire file, sheets are processed in the order listed on the Control Sheet, from left to right.
- Within a sheet, by default Axiom queries are processed sequentially in ascending order by Axiom query number (AQ1, then AQ2, and so on.). If batch numbers are used, then processing works as follows:
    - If a batch number is defined for an Axiom query, then all queries with the same batch number are processed *in parallel* before any other Axiom queries on the sheet. If multiple batch numbers are specified, the batches are processed in ascending order (batch 1, then batch 2, and so on).
    - After the batch queries are processed, the remaining queries on the sheet are processed sequentially in ascending order as normal.

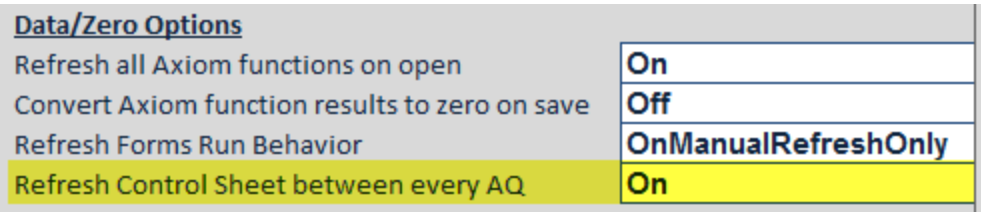
  For more information on using batch processing, see Batch processing for Axiom queries.

It is important to understand this order when setting up configurations such as nested queries (where one query is built out by a prior query), or configurations where the settings of one query are impacted by the results of a previous query.

Refreshing Axiom query settings between queries

By default, when a file is refreshed, all of the Axiom query settings are read once at the start of the process. If a query setting is dynamic and changes during the processing, that change will not be recognized until the next time the file is refreshed.

If you want an Axiom query setting to be dependent on the results of another Axiom query, you can modify this behavior so that the Control Sheet is calculated after each query is run, and each query's settings are read individually before the query is run. To do this, set **Refresh Control Sheet between every AQ** to **On**. This setting is located on the Control Sheet, in the **Data/Zero Options** section.



For example, you might enable this setting if:

- You want one query to define a data filter for a subsequent query.
- You want a query to be enabled or disabled based on the results of a prior query.

You should only enable this option if you have query settings that are dependent on the results of other queries. Enabling this option can introduce many additional calculation cycles to the refresh process, which may impact performance.

Data security considerations

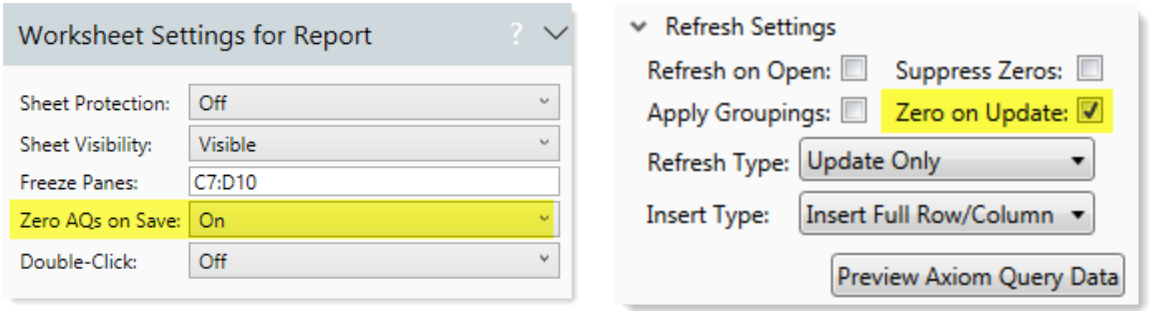
Each user's table filters from security are applied when an Axiom query is refreshed. However, before the refresh occurs, the file will display whatever data was last-saved in the file, such as from a previous user refreshing the query and then saving the file.

To prevent users from seeing Axiom query data that was inadvertently saved in a report, you should configure the file using one (or both) of the following options:

- **Zero AQs on Save**: If this option is enabled, then when the file is saved the Axiom query data is converted to zeroes. The next user to open the file would see zeroes until a refresh occurs.

  This option can be configured on a per sheet basis using the Sheet Assistant, in the **Worksheet Settings** section. On the Control Sheet, it is located in the Data/Zero Options section, and is named **Convert Axiom Query results to zero on save**.

  If the refresh behavior for the query is **Update Only** or **Insert and Update**, then you must also enable **Zero on Update** in order to update the data with zeroes on save.

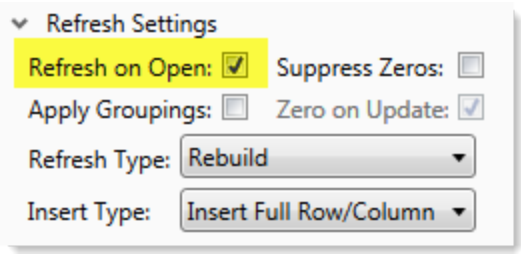


NOTES:

- The refresh behavior **Refresh during document processing** must be enabled in order for an Axiom query to zero on save. This behavior is enabled by default.
- If some Axiom queries use the refresh behavior **Refresh after save data**, then the queries will be zeroed *before* the file is saved. After the file has been saved, the queries will be refreshed.
- If an Axiom query has an assigned batch number, that query is not zeroed on save.
- If an Axiom query uses **Insert Only** refresh behavior, then the query will be zeroed on save but that zeroed data will never be updated. Either the zero on save option should not be enabled for the sheet, or the query should use a different refresh behavior (such as **Rebuild** or **Insert and Update**).

- **Refresh on Open**: If this option is enabled, then when the file is opened the Axiom query automatically refreshes (and therefore the user's table filters are applied). In this case it doesn't matter if the prior user's data was saved within the file, because the data will be immediately replaced with the current user's data when the file is opened.

  This option can be configured on a per Axiom query basis using the Sheet Assistant, in the Axiom Queries > Refresh Settings section. On the Control Sheet, it is located in the Axiom query settings, in the Refresh behavior section.



This consideration does not apply to plan files in most cases. Generally, plan files are designed to be refreshed by an administrator (or other power user) and then that data is saved within the file for planning purposes. Access to the plan file is then restricted to only those users who need to work with that data.

Previewing Axiom query data

Once you have configured an Axiom query, you can preview the data to be returned, using the **Preview Axiom Query Data** button on the Sheet Assistant.

The preview is a flat view of the data as it exists in the database. The intent of the preview is to confirm that you are querying the intended set of data, at the intended sum level, before it is inserted into a sheet. For example, if the query returns 200 records of data when you are only expecting 50, then this may mean that the sum level of the query needs to be adjusted, or that the data filter is not set as intended.

The entries in your field definition are used to determine the columns of data returned. Column filters and other special field definition settings are honored for this purpose.

NOTE: If the Axiom query uses a system table as the primary table (such as Axiom.Aliases), then the preview displays all columns of data in the table. Field definition row entries are ignored.

Only settings that impact the data query itself are honored by the preview. For example, the Axiom query data filter is applied, but any data range filters are not.

Axiom query settings

Axiom queries are defined on the Control Sheet of any Axiom file, on a per sheet basis. Settings must be entered in the column for the applicable target sheet.

Each sheet can have multiple Axiom queries. The default Control Sheet has settings available for one Axiom query. If you need more Axiom queries, you can add more using the Sheet Assistant.

Keep in mind that you can use multiple data ranges for the same Axiom query. If the Axiom query settings are the same but you simply need to group data into multiple sections, you can use data range filters instead of defining multiple Axiom queries. For more information, see Creating the insert control column or row for an Axiom query.

File-wide Axiom query settings

The following Axiom query option applies to all sheets in the file. This option is located in the **Workbook Options** section of the Control Sheet.

Field	Description
Process cross sheet AQ Batches	Specifies whether Axiom query batches are processed across sheets or per sheet. By default this setting is **Off**, which means that any batched Axiom queries are processed per sheet. If set to **On**, then batched Axiom queries are processed across sheets. This means that queries from different sheets can be processed concurrently in the same batch. For more information, see Batch processing for Axiom queries.

Sheet-wide Axiom query settings

The following options apply to all Axiom queries on a sheet, not individual Axiom queries. These options are located in the **Data/Zero Options** section of the Control Sheet.

Item	Description
Convert Axiom Query results to zero on save	Specifies whether Axiom query data is zeroed when the file is saved. The default setting is **Off**. This setting is configured on a per sheet basis and applies to all Axiom queries on the sheet. This feature is intended for data security within reports. If an Axiom query is *not* set to refresh on open, then the data queried and saved by one user will still be visible when another user opens the file. This may result in users seeing data outside of their data filter, because the query will not update until the user explicitly refreshes the file. If this option is set to **On**, then when the file is saved, all Axiom queries in the sheet are zeroed as follows (depending on the refresh behavior for the query): • If the query is set to rebuild, then all rows in the query are deleted, and the data ranges are collapsed. • If the query is set to update and/or insert, and zero on update is not enabled, then no action occurs. The rows in the data ranges are left as is. • If the query is set to update and/or insert, and zero on update is enabled, then the existing rows are left in the data range, but the data values are zeroed. For more information on which columns are zeroed, see the discussion on zero behavior in Specifying how data is refreshed in an Axiom query.

Item	Description
	NOTES: • The refresh behavior **Refresh during document processing** must be enabled in order for an Axiom query to zero on save. This behavior is enabled by default. • If some Axiom queries use the refresh behavior **Refresh after save data**, then the queries will be zeroed *before* the file is saved. After the file has been saved, the queries will be refreshed. • If an Axiom query has an assigned batch number, that query is not zeroed on save. • If an Axiom query uses **Insert Only** refresh behavior, then the query will be zeroed on save but that zeroed data will never be updated. Either the zero on save option should not be enabled for the sheet, or the query should use a different refresh behavior (such as **Rebuild** or **Insert and Update**).
Enable full AQ query validation mode	Specifies whether all AQ data range filters are sent to the server for parsing as part of the query to the database, or just the first data range filter. This helps determine the data to be returned by the query. • By default this is set to **Off**, which means that only the first data range filter is sent to the server. In the majority of cases, this is sufficient to ensure that the data returned by the data query will be compatible with all data range filters in the report. • If **On**, then all data range filters for the query are sent to the server. This may be necessary in cases where the data range filters specify very different sets of data. Enabling this option will increase the complexity of the data query statement and therefore may slow report performance. It is not recommended to enable this option unless it is necessary for the data query. Contact Axiom Support if you need assistance determining whether this option should be enabled.

Item	Description
Refresh Control Sheet between every AQ	Specifies whether Axiom query settings are read once at the start of the refresh process, or if they are read before running each individual Axiom query. • By default this is set to **Off**, which means that all Axiom query settings are read at the start of the refresh process. In this scenario, Axiom queries cannot be dependent—meaning, the results of one Axiom query cannot affect the settings for another Axiom query. • If **On**, then the Control Sheet is calculated after each Axiom query is run, and the individual query settings are read before each query is run. This option allows for dependent queries. The results of one query can impact the settings of a subsequent query. For example, one of the initial queries might determine the data filter for a subsequent query, or the results might impact whether a subsequent query is active or not. The Control Sheet is always refreshed after all Axiom queries have been run, to update all Control Sheet settings before reapplying features such as freeze panes. **NOTE:** You should only enable this option if you have dependent Axiom queries. Enabling this option can introduce many additional calculation cycles to the refresh process, which may impact performance.

General Axiom query settings

Each Axiom query has the following general settings:

Item	Description
Name	The name of the Axiom query. This name is required for templates/plan files, and is optional for all other files. If a query has a name, the name is used whenever lists of Axiom queries are displayed. The name should be unique and descriptive. When processing plan files, you are prompted to choose which Axiom queries to update by name.
Activate	Determines whether the Axiom query is active. • If **On**, the query is active and can be run. • If **Off**, the query is inactive and cannot be run. EXCEPTION: The RunAxiomQueryBlock function does not require the target Axiom queries to be active.

Query details

The query details define the general parameters of the data query.

Item	Description
Primary Table	Specifies the primary table to query data from when refreshing the file. In most cases, the primary table is a data table (such as GL2020), but it can be any table. For example, the primary table could be a reference table such as DEPT, if you wanted to create a list of departments or department attributes. Axiom queries can query data from multiple tables. The primary table setting simply specifies the base table for the query. If the primary table is not specified, Axiom determines the primary table based on the first data column in the field definition. This is primarily intended for situations where you are using column alias names in the field definition, and the table that the alias names point to may change over time. For more information, see: • Specifying the primary table for an Axiom query • Using multiple tables in an Axiom query
Sum data by these columns	Defines the level at which to sum or group the data returned by the query. You can enter multiple column names, separated by a semi-colon. You can use full Table.Column syntax or "column-only" syntax. If the table name is omitted, the primary table for the Axiom query is assumed, except in cases where the column is a validated column, in which case the lookup table is assumed. For example, an entry of `DEPT` will be interpreted as `DEPT.DEPT`. However, if you want to sum by `DEPT.Region`, the full Table.Column syntax is required. If this setting is left blank, then by default the data is summed using the lookup columns for the key columns of the primary table. For more information, see Defining the sum level for an Axiom query.

Item	Description
Sort by database columns	Specifies the database columns to sort the data by, prior to inserting it into the target sheet. This setting does not apply when only updating data. You can enter multiple column names, separated by a semi-colon. You can specify ascending (asc) or descending (desc) after each field name. The default is ascending if nothing is specified. For example: `Dept asc; Acct desc` You can use full Table.Column syntax or "column-only" syntax. If the table name is omitted, the primary table for the Axiom query is assumed, except in cases where the column is a validated column, in which case the lookup table is assumed. For example, an entry of `DEPT` will be interpreted as `DEPT.DEPT`. If this setting is left blank, and no spreadsheet sort is specified, then the data is sorted by the columns specified in the **Sum data by these columns** setting, in ascending order. For more information, see Defining the sort for an Axiom query.
Sort results by these columns	Specifies the spreadsheet columns to sort the data by when the query is refreshed. You can enter up to three column letters, separated by a semi-colon. You can specify ascending (asc) or descending (desc) after each column letter. The default is ascending if nothing is specified. For example: `C asc; D desc` For more information, see Defining the sort for an Axiom query.

Item	Description
Data Filter	Defines a data filter to limit the data query. For example: `Dept=1000` or `ACCT.AcctGroup='Benefits'`. You can use full Table.Column syntax or "column-only" syntax. If the table name is omitted, the primary table for the Axiom query is assumed, except in cases where the column is a validated column, in which case the lookup table is assumed. For example, an entry of `DEPT` will be interpreted as `DEPT.DEPT`. If this setting is left blank, the Axiom query returns all records in the specified table, summed by the specified sum level. Therefore, a filter is almost always used when setting up a query for a template, so that the data in the resulting plan files are limited by the appropriate plan code. For reports, a data filter is optional. Note that a user's table access settings (as defined in Security) are always applied to an Axiom query, in addition to any data filter. For more information, see Defining a data filter for an Axiom query.
Limit query data based on another table	If desired, you can limit the data in this Axiom query based on the results of another query. For example, you may want to only include items in this query if those same items are also found in a different table. This is an advanced option. Special syntax is used to identify the parameters of this "pre-query". For more information, see Limit the data in an Axiom query based on another query.
Post Query Filter	If desired, you can define a filter to limit the data to be returned by the query. This filter is applied after the initial database query, but before data is returned to the client (so it is still a database-level query, unlike data range filters). This is an advanced option. The primary use of this feature is to enable filtering based on segment when segmenting your query data; however it can be used for other purposes. For more information, see Organizing Axiom query data into ranked segments.
Post Query Sum By	This is an advanced option that only applies when segmenting your query data. You can list the name of the segment column here, in order to group the query results by segments. For more information, see Organizing Axiom query data into ranked segments.

Item	Description
Suppress records with zero values	Specifies whether rows with all zero values are inserted into the sheet. After the data query, but before inserting data into the sheet, Axiom checks the data to determine if any records contain all zero values. If this setting is enabled, zero value records are not inserted into the sheet. Non-key columns that meet both of the following criteria are evaluated to determine whether a row should be hidden: • The column data type is Integer (all types) or Numeric. • The column is from the primary table or an additional data table. If the primary table is a data table, Integer and Numeric columns on lookup reference tables are ignored—meaning these columns may have values, but the row is still suppressed if all applicable data table columns have zero values. There is one exception: reference table columns are considered if the column classification is Values and the numeric type is Currency.
Max row warning threshold	Specifies the maximum number of records that can be returned by an Axiom query without triggering a warning. You can leave this blank to use the system configuration setting, or you can enter a number to set the threshold for this particular query. This setting is intended to prevent users from accidentally bringing in an extremely large number of records, potentially resulting in excessive processing times or system errors. If the number of records in the query exceeds this setting, the user is warned of this and asked if they want to continue. **NOTES:** • The only user options are to cancel the query or continue with the full number of records. This setting defines the threshold for the warning, it does not limit the query to the specified number of records. • The warning only applies when users manually refresh Axiom queries. If the query is processed by Scheduler, the processing will fail if the query exceeds the threshold. For more information on defining system configuration settings, see the *System Administration Guide*.

Item	Description
Limit query to top "n" results	If desired, you can limit the query results to the top N records for the query—such as top 5, top 10, or top 50. Enter the number of records that you want returned. When using this setting, the database sort order of the query is very important as that will determine which records are returned. For more information, see Return the top N records for an Axiom query.
Process Definition ID	Specifies the process definition ID to use when querying plan file process columns. You can use the GetProcessInfo function to return the ID for a process definition, or you can hover over the definition in Axiom Explorer or the Explorer task pane to read the ID on the tooltip. When the primary table of an Axiom query is the plan code table of a file group, you can optionally include process columns in the query to return process information for each plan code.
Batch Number	Assigns the query to a batch for parallel processing. If a batch number is specified, then all queries in the sheet with the same batch number will be processed in parallel instead of sequentially, before any non-batched queries are processed. For more information, see Batch processing for Axiom queries. If the batch number is blank, then the query does not belong to a batch and will be processed normally. For more information, see Processing order of Axiom queries.

▶ Display / user interaction options

The following options relate to how data is displayed for the query, and how users may interact with the data.

Item	Description
Insertion Behavior	Specifies the insertion behavior for the query, when records are inserted into the data ranges. • **Insert** (default): New records are inserted by inserting rows or columns within the data ranges. • **InsertRange**: New records are inserted by inserting cells within the data ranges. If InsertRange is selected, then you must specify the insertion range using [Stoprange] and [Startrange] tags. The [Startrange] tag is optional; if omitted, the insertion range starts at the [AQ#] tag. The insertion range tags must be placed in the same column or row as the [AQ#] tag (depending on the query orientation). This option does not apply if the refresh behavior for the query is set to update-only. For more information, see: • Insertion options for Axiom queries • Defining the insertion range when using Insert Range
Apply grouping to data ranges	Specifies whether spreadsheet grouping is applied to the Axiom query results. If **Off**, grouping is not applied and the results are presented as normal. If **On**, grouping is applied and the query results are grouped by data ranges. The row containing the [AQ#] tag is used as the "anchor" row for the grouping. All rows in the range, including the row with the [Stop] tag, are contained in the group. This option is most typically used in cases where you have multiple data ranges, or multiple nested Axiom queries, and you want to group sections in the results so that they can be expanded and collapsed. **NOTES:** • Grouping is only supported for use with standard vertical queries. This setting is ignored for horizontal queries. • Only six Axiom queries can be enabled for grouping per sheet. For more information, see Using spreadsheet grouping with an Axiom query.

Item	Description
Change orientation to horizontal	Specifies whether the query populates data vertically or horizontally. If **Off** (the default), the query is a standard vertical query. If **On**, the query becomes a horizontal query. Certain settings, such as control rows and columns, differ based on the orientation of the query. You will complete either the **Vertical Configuration** section or the **Horizontal Configuration** section, depending on the orientation of the query. All other settings apply to both query orientations, unless otherwise noted. For more information, see: • About Axiom queries • Using horizontal Axiom queries
Drillable	Specifies whether the query is enabled for drill-down and drill-through drilling. By default, this is set to **On**, and the query rows can be drilled if the query is otherwise eligible for drilling. If **Drilling** is disabled, then the query is not an eligible context for drilling. You might do this for either of the following reasons: • The query does not make sense to drill, even though it may be technically eligible for drilling. For example, many Axiom queries in templates/plan files do not make sense to drill, because their data filter restricts them to a single department, and the rows are already at the lowest level of detail. • The data in the query may have an alternate drill context, and you want to make that context available for drilling. For example, if you use an Axiom query to build out a section of GetData functions, the GetData functions cannot be drilled down because the drilling context is the Axiom query. However, if you disable drilling for the Axiom query, then users can drill down the GetData functions as if they were unassociated with an Axiom query. **NOTE:** This setting does not apply to horizontal queries, because horizontal queries are not eligible for drilling.

Vertical configuration

The following Control Sheet settings apply if the query is a standard vertical query. This section defines the location of various control rows and control columns on the sheet, and the calc method options.

NOTE: If you are using the Sheet Assistant to complete query settings, the Sheet Assistant adjusts to only show the relevant settings for the current orientation.

Control rows and columns

Each vertical Axiom query requires the following control rows and columns to be defined. You can use the default settings in the Control Sheet, or specify different rows and columns.

Item	Description
Field Definition Row(s)	Specifies the row or rows in the target sheet where database column names are placed. Data from the specified database column will be brought into the sheet column (within the Axiom query's data range) when the file is refreshed. To specify the row, you can enter a single row number (1), or you can enter the row or rows as a range (1:1 or 10:12). When entering a row range, make sure that Excel is evaluating the range as text. If the cell is not formatted as text, you can force a text entry by entering an apostrophe in front of the range ('10:12). For more information, see Creating the field definition for an Axiom query.
Insert Control Column	Specifies the column where data range tags are placed. These tags define the beginning and end of each data range. An Axiom query can have multiple data ranges, to determine where data rows are inserted into the sheet. **NOTE:** By default, this column also serves as the data control column for the Axiom query. When the query is refreshed and the data range is populated, the cells in between the data range tags will contain the key codes for each row. For more information, see Creating the insert control column or row for an Axiom query.
Internal Data Control Column	Optional. Specifies a separate column to be used as the data control column. By default, the insert control column is automatically used as the data control column. The data control column holds the key codes for each row of data, to be used for subsequent data updates (if applicable). The primary purpose of this separate setting is for backward-compatibility when upgrading older Control Sheets—previous versions of Axiom required a separate column. For more information, see About data control codes for Axiom queries.

Calculation methods

A vertical Axiom query can use either an in-sheet calc method row or a calc method library. If an in-sheet calc method row is specified on the Control Sheet, that row will be used when the Axiom query is processed, regardless of any calc method library settings. If you want to use the calc method library, make sure that the **In-Sheet Calc Method Row(s)** setting is blank. For more information, see Calc methods and Axiom queries.

Calculation methods only apply if the refresh behavior is rebuild or insert.

In-sheet calc method settings

Item	Description
In-Sheet Calc Method Row(s)	Specifies the row or rows in the target sheet where formatting and calculations are defined. The formatting and formulas in this area are applied to data rows as they are inserted into the sheet. The number of rows in the in-sheet calc method must be equal to or greater than the number of rows in the field definition. For example, if the field definition is two rows, the in-sheet calc method must be at least two rows. This setting is required for reports. If you are working in a template, for any particular Axiom query you can choose to use either the calc method library or an in-sheet calc method row. If you want to use the calc method library, you must leave this item blank. If all calc method settings are blank, then the field definition row is assumed as the in-sheet calc method row. For more information, see Using in-sheet calc methods with Axiom queries.

Calc method library settings (file group files only)

Item	Description
Assign from data field	Specifies the database table and column where calc method assignments have been defined. For example, `ACCT.AssignCM`. This setting only applies to templates/plan files. Special syntax is available if you need two levels of lookup in order to assign calc methods (for example, if the calc method used for an account varies based on the department type). For more information, see: • Using a single assignment column • Using multiple assignment columns by plan code type

Item	Description
Assign from sheet	Use this section if you want to read the calc method assignments from a sheet in the file, rather than pointing directly to a database column. Within the sheet, you can use any methodology to create the list of assignments, including using another Axiom query to build the list dynamically per plan code. Complete all four settings as follows: • **Key column name**: The name of the key column for the calc method assignments. For example, ACCT. • **Sheet Name**: The name of the sheet that contains the calc method assignments. For example, CMAssign. • **Key column in sheet**: The column in the sheet that contains the key codes. For example, enter C if the account codes are in column C. • **CM Assignment column in sheet**: The column in the sheet that contains the calc method assignments. For example, enter D if the assignments are in column D. For more information, see Reading calc method assignments from a sheet.
Default Calc Method	Specifies the default calc method to use if there is no assignment for a particular plan code. This setting only applies to templates/plan files. The default calc method is not applied if an assignment is invalid—instead, an error results. For more information, see Using calc method libraries with Axiom queries.

Horizontal configuration

The following Control Sheet settings apply if the query is a horizontal query. This section defines the location of various control rows and control columns on the sheet, and the calc method options.

NOTES:

- By default, all queries are vertical queries. If you want to define a horizontal query, then you must change the setting **Change orientation to horizontal** to **On**.
- If you are using the Sheet Assistant to complete query settings, the Sheet Assistant adjusts to only show the relevant settings for the current orientation.

Control rows and columns

Each horizontal Axiom query requires the following control rows and columns to be defined. You can use the default settings in the Control Sheet, or specify different rows and columns.

Item	Description
Field Definition Column(s)	Specifies the column or columns in the target sheet where database column names are placed. Data from the specified database column is brought into the sheet row (within the Axiom query's data range) when the file is refreshed. To specify the column, you can enter a single column letter (D), or you can enter the column or columns as a range (D:D or D:F). When entering a column range, make sure that Excel is evaluating the range as text. If the cell is not formatted as text, you can force a text entry by entering an apostrophe in front of the range ('D:F). For more information, see Creating the field definition for an Axiom query.
Insert Control Row	Specifies the row where data range tags are placed. These tags define the beginning and end of each data range. An Axiom query can have multiple data ranges, to determine where data records are inserted into the sheet. **NOTE:** By default, this row also serves as the data control row for the Axiom query. When the query is refreshed and the data range is populated, the cells in between the data range tags will contain the key codes for each column. For more information, see: • Creating the insert control column or row for an Axiom query • About data control codes for Axiom queries

Calculation methods

Calculation methods only apply if the refresh behavior is rebuild or insert. For more information, see Calc methods and Axiom queries.

Item	Description
In-Sheet Calc Method Column(s)	Specifies the column or columns in the target sheet where formatting and calculations are defined. The formatting and formulas in this area are applied to data records as they are inserted into the sheet. The number of columns in the in-sheet calc method must be equal to or greater than the number of columns in the field definition. For example, if the field definition is two columns, then the in-sheet calc method must be at least two columns. If this setting is left blank, then the field definition column is assumed as the in-sheet calc method column. For more information, see Using in-sheet calc methods with Axiom queries.

Refresh behavior

Refresh behavior determines how rows are updated when the file is refreshed. For more information, see Refresh and insertion behavior for Axiom queries.

Item	Description
Refresh on manual refresh	If enabled, the query will be refreshed when a user manually refreshes the sheet. For example, the user could click the **Refresh** button in the ribbon, or click a task pane item that is configured to perform a refresh, or press F9. This option is enabled by default. If you disable this option, then the query will not refresh when a user manually refreshes the sheet. For example, you might disable this option if you have a query that you *only* want to run when the file is opened, and not during any subsequent manual refreshes. **NOTE:** In plan files only, users cannot manually refresh Axiom queries unless they have the **Run AQs in Plan Files** permission. If you have a query that needs to be run in plan files, but you do not want to give users this permission, then you can enable **Refresh on file open** for the query.
Refresh during document processing	If enabled, the query will be refreshed when the file is processed by a feature that includes performing a refresh, such as Process Plan Files, File Processing, or Process Document List. Some of these features, such as Process Plan Files, allow you to disable refreshing particular queries as part of that particular processing job. For other features, the query will always be refreshed if this option is enabled. This option is enabled by default. If you disable this option, then the query will not be available to processing features that include performing a refresh.
Refresh during template processing	If enabled, the query will be refreshed when the file is processed by the Scheduler task Process Template List. This option only applies to file group template files; it has no effect in any other file type. When the query is processed by Process Template List, the **Last refresh time** of the query is updated in the template. This feature is intended to enable use of time-stamped Axiom queries in virtual plan files, by time-stamping the query in the template.

Item	Description
Refresh on file open	If enabled, the query will be refreshed automatically whenever the file is opened, whether by a user or by a system process (with one exception: Process Plan Files—see notes below). This setting is disabled by default. When this setting is enabled, administrators can use the **Open Without Refresh** option if you need to open the file without running the Axiom query. For more information on this option, see Opening an Axiom file without refreshing data. **NOTES:** • If the file uses refresh forms (refresh variables or Axiom form as refresh form), the refresh form will only display on file open if you explicitly configure it to do so using the **Refresh Forms Run Behavior** setting on the Control Sheet. • In plan files, queries configured to refresh on open are always run when the file is opened by a user, regardless of whether the user has the **Run AQs in Plan Files** security permission. This can be used if you have a query that needs to be run to enable functionality in the plan file (such as for a drop-down list), but you do not otherwise want users to be able to refresh Axiom queries. • This setting is ignored by Process Plan Files. However, the query can still be selected for processing if it is also enabled to **Refresh during document processing**.
Refresh once before multipass processing	This option only applies when the file is processed using the multipass option of File Processing. If enabled, the query will be run once before multipass processing begins. If you do not want the query to also be run during each individual pass, then you should disable **Refresh during document processing**. This option is disabled by default. You might enable this option if you have a query that needs to be run to bring an initial set of data into the file, but you don't want to set the query to refresh on open because the query is quite large and causes an unnecessary delay when opening the file for editing.

Item	Description
Refresh after save data	If enabled, the query will be refreshed automatically after a save-to-database occurs, whether the save is initiated by a user or by a system process. The refresh occurs regardless of whether any data changes were actually saved. The Axiom query is refreshed after *all* save-to-database processes in the workbook are completed. This setting is disabled by default. You might enable this option if you have a configuration where the save-to-database process adds or updates a record that would be returned by an Axiom query in the workbook. In order for the data in the Axiom query to be updated automatically for the changed data, you would need to enable this option (otherwise the user would not see the changed data until they manually refresh). This feature may be especially useful in Axiom form design. **NOTE:** The behavior of this option depends on whether the sheet-level setting **Convert Axiom Query results to zero on save** is also enabled. If "zero on save" is enabled, then the Axiom query is first zeroed, then the file is saved, then the Axiom query is refreshed with data. This order is so that the data is not saved within the file. However, if "zero on save" is *not* enabled, then the Axiom query is refreshed before the file is saved, which means that the current data will be saved within the file. This does not apply when interacting with Axiom forms, because in that case the file is never saved.
DataLookups to run	Specifies one or more data lookup queries to run after the Axiom query is refreshed, by listing the names of the DataLookup data sources. Separate multiple data source names with commas. Multiple data sources will be run in the order specified. The list of data source names can be qualified with sheet names (such as `Sheet1!Data`) or unqualified (just `Data`). If qualified, then Axiom will only scan the listed sheets for the data sources. If unqualified, then Axiom must scan all sheets for the data sources, which can impact performance. All items in the list must be either qualified or unqualified. Mixing qualified and unqualified names will result in an error when the data sources are executed. All unnamed DataLookup data sources will also be run when any Axiom query is refreshed. For more information, see Executing data lookups.

Item	Description
Refresh only if primary table changed since last refresh	If enabled, the Axiom query will only refresh if dependent tables have been modified since the query was last executed (also known as time-stamped refresh behavior). If none of the dependent tables have been modified, then the query will not be run and the data in the sheet will be left as is. This is a performance optimization feature that is intended for queries that meet specific criteria. For more information, see Refresh data only when dependent tables have been modified (time-stamped Axiom queries). The dependent tables for the query are the primary table and any additional tables listed in the **Additional table dependencies** field. The **Last refresh time** is used to determine when the query was last executed.
Additional table dependencies	Specifies one or more additional tables to consider if **Refresh only if primary table changed since last refresh** is enabled. Separate multiple table names with commas. If left blank, then the primary table is the only dependent table for the time-stamped refresh behavior.
Last refresh time	The date and time the query was last executed. This value is automatically written to this field when the query is run. Currently, this value is only used if **Refresh only if primary table changed since last refresh** is enabled. Otherwise, it is for information only. The last refresh time is shown in UTC (Universal Coordinated Time); it is *not* converted to the local time of the client where the refresh occurred. This field is system-controlled and should not be manually modified. The only exception is that occasionally you may need to clear this field in order to "reset" the last refresh time. **NOTE:** By default, this field is not populated when executing an Axiom query in a file group template file. This is so that when plan files are created from the template, the field starts out blank and can be populated for the plan file. However, if the Axiom query is refreshed by Scheduler using the Process Template List task, then the last refresh time is populated.
Rebuild Data Ranges	Determines whether Axiom queries are rebuilt on refresh. If **On**, all content in the data range is deleted when the file is refreshed, and the data ranges are rebuilt. In this case, the insert, update, and zero settings are ignored. If **Off**, the existing content in data ranges is retained, and data is updated according to the other refresh behavior settings.
Zero data before update	Determines whether data values previously inserted by the Axiom query are zeroed before subsequent updates. This setting only applies if **Rebuild Data Ranges** is **Off**.

Item	Description
Update existing data	Determines whether data values previously inserted by the Axiom query are updated on subsequent refresh. Formatting and formulas are not updated. This setting only applies if **Rebuild Data Ranges** is **Off**.
Insert new rows	Determines whether new lines are inserted on refresh. This setting only applies if **Rebuild Data Ranges** is **Off**.

Data conversion

You can apply data conversion to an Axiom query, if the table being queried has been configured for data conversion. For example, you might use these settings to convert the data being queried from U.S. dollars into Canadian dollars. For more information, see Querying data using data conversions.

Item	Description
Conversion Target	Specifies a conversion target so that the Axiom query returns converted data. This is the only required entry for data conversions. You can enter a fixed value that will apply to all records, or you can look up the value on a record by record basis from a grouping column in a lookup reference table. The fully qualified name must be used—for example, `DEPT.TargetCur`. **NOTE:** If you use a grouping column, it is recommended to also include the grouping column in the field definition for the query, to identify the conversion that was applied for each individual record. In either case, the conversion target must be a value that is found in the "to" column for the associated conversion table. The conversion table is determined on a column by column basis (for each data column entry in the field definition), using the conversion configuration for the associated table. If a table in the Axiom query is not configured for data conversions, then the conversion settings are ignored and the "as is" data is returned for that table (unless all tables in the query are not configured for data conversions, in which case an error results).
Conversion Scenario Override	If desired, specify a conversion scenario to override the default scenario for the data conversion. The entry must be a value that is found in the "scenario" column for the conversion table. The conversion table is determined based on the primary table for the query.

Item	Description
Conversion Type Override	If desired, specify a conversion type to override the default type for the data conversion. The entry must be a value that is found in the "type" column for the conversion table. The conversion table is determined based on the primary table for the query.

If any of these entries are invalid, the conversion results will be zero values. For example, if you specify EURO as the conversion target, but the actual code in the "to" column for the conversion is EUR, then all values subject to the data conversion will return zero. Data conversions will also return zero values if the conversion configuration for the table has invalid values, or if the conversion table itself has no valid rate entry for the relevant conversion and period (for example, blank entries in a rate column).

5

Data Lookups

Data lookups are data queries that can be configured to execute at specific times. They can serve as a regular data refresh option, or they can be used to bring in "static" points of data that do not need to be continually refreshed while the user is working in the file.

Data lookups are created by using a DataLookup data source. Each row in the data source defines the parameters for a value that you want to return from the database. When the query is executed, the value for each row is returned into a designated column of the data source. These values can then be referenced in the file just like values returned via Axiom functions. Currently, data lookups can be used to return the same data as the following Axiom functions:

- GetData
- GetFeatureInfo
- GetFileGroupProperty
- GetFileGroupVariable
- GetFileGroupVariableEnablement
- GetFileGroupVariableProperty
- GetPlanItemValue
- GetSecurityInfo
- GetTableInfo
- GetUserInfo

Data lookups support a variety of execution options to meet various needs. They can be configured to run automatically whenever a file is refreshed, or they can be configured to run only at specific times—such as on file open, or after a particular Axiom query is executed. You can also optionally expose the ability for end users to execute data lookups as needed, using the **Execute Data Lookups** command.

To configure a file to use data lookups, you must:

- Define one or more DataLookup data sources in the file.
- Configure the data lookup execution options as needed, so that the query executes when you want it to.

When to use data lookups

Data lookups are primarily intended to be a substitute for Axiom functions. For example, plan files typically need to query certain "static" points of data, such as the current plan code and description, which are then referenced throughout the file. Once a plan file is opened and these values are queried from the database, it is not necessary to refresh the values again during the current file session, because the plan code and description will not change. If you use Axiom functions to return the values, the functions will continue to execute a data query each time the file is refreshed or each time the cell is forced to calculate. However if you use a data lookup to return the value, then the query can be configured to only execute when you tell it to—in this example, when the file is initially opened.

Whenever possible, data lookups should be used instead of Axiom functions to bring in this type of data, for performance reasons. Data lookups have the following advantages over Axiom functions:

- Execution of data lookups can be tightly controlled, unlike the behavior of Axiom functions. Axiom functions are updated each time the file is refreshed, and also any time the cell is forced to calculate using standard spreadsheet calculation behavior. If the values returned by these functions are not expected to change, then this processing overhead is unnecessary and can be eliminated to help improve file performance.
- The database query used by data lookups is more efficient than Axiom functions, greatly reducing the database traffic required to return each value.

In some cases it is also possible to use a "refresh on open" Axiom query to return the desired data. For example, you can use an Axiom query to return values from a driver table, where the driver values are unlikely to change during the current file session. If the "manual refresh" behavior is disabled for the query, then the query will only run once when the file is opened; it will not execute again on subsequent refreshes. If it is possible to return the data via Axiom query, then that is generally the recommended approach.

However, sometimes you need to return a set of disparate data points that cannot be easily returned using an Axiom query, because the data cannot be joined. In this case, data lookups provide a flexible alternative. Each row in the DataLookup data source can query any type of data; the rows do not have to be related. One way to think of it is that each row in the data source corresponds to an Axiom function and can return any data that the Axiom function can.

Creating DataLookup data sources

To create a data lookup query, use a DataLookup data source. Each row in the data source defines the parameters for a value that you want to query from the database.

The DataLookup data source supports several different row types that can be used to query different types of data. For example, you can use a `[GetData]` row to query data from any client-defined table, or a `[GetFileGroupProperty]` row to return information about a file group. You can mix row types in the same data source as needed.

Each sheet in the file can have multiple DataLookup data sources. The parameters on a row in one data source can depend on values returned by another data source—much like the parameters of one function can depend on values returned by other functions. However, when using dependent data sources, you must make sure that the data sources are executed in the appropriate order.

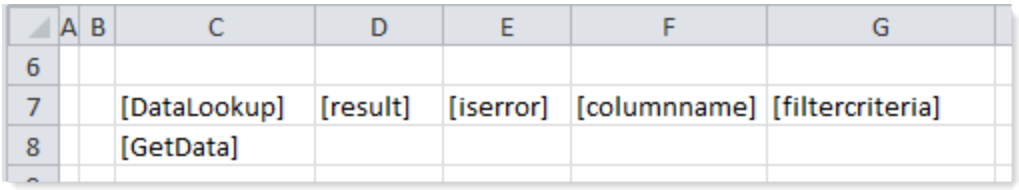
To create a DataLookup data source:

- Right-click the cell in which you want to start the data source, then select **Axiom Wizards > Insert Data Lookup Data Source > Insert *TagName***.

  The tag name specifies the type of row tag that you want to add to the data source. For example, if you want to use a `[GetData]` row, you would select **Insert Get Data Tag**. You can use multiple different tags in each data source, but you must choose one to start with.

  The current cell and any necessary cells below and to the right must be blank in order to insert the data source.

The wizard adds the primary tag, all required column tags for the selected row tag, and one row tag. The row containing the DataLookup tag becomes the control row, and the column containing the DataLookup tag becomes the control column. The following screenshot shows an example data source after the initial insertion, with no parameters defined yet:



	A	B	C	D	E	F	G
6							
7			[DataLookup]	[result]	[iserror]	[columnname]	[filtercriteria]
8			[GetData]				

From here, you can complete the data source as follows:

- You can optionally name the data source and specify an order. This is done by adding parameters to the primary `[DataLookup]` tag. To do this, you must manually edit the tag using the appropriate syntax. For more information, see DataLookup tag syntax.
- You can manually complete the parameters for the row tag by filling out the cells as appropriate, or you can use the Data Source Assistant to complete the parameters. The parameters vary depending on the type of row you are defining. For more information on the row types and column parameters, see Defining data lookup rows and columns.
- You can add more rows to the data source manually, or you can use the Data Source Assistant to add rows. For more information, see Using the Data Source Assistant to add or edit data lookups.

NOTE: Once you have set up the data source, remember to configure the data lookup execution options as needed so that the query is run when you expect. If the query is unnamed, you may not need to do anything further, but if the query is named then you must configure a way to trigger the execution. For more information, see Executing data lookups.

DataLookup tag syntax

The primary DataLookup tag uses the following syntax:

`[DataLookup;DataSourceName;Order]`

Parameter	Description
DataSourceName	Optional. The name of the data source. For example: `[DataLookup;FileGroup]` The name of this data source is FileGroup. You can reference this name when using features that list data lookups to execute. A data source should only be left unnamed if you want it to be executed every time the file is refreshed. If instead you want to control the execution to only certain times, then you should give the data source a name. For more information, see Executing data lookups. Multiple data sources can have the same name. If you list that name to be executed, all eligible data sources with the same name will be run. The reserved name `AxRefreshOnOpen` can be used if you want the data source to be automatically executed when the file is opened. If you use this name, it is not necessary to list the data source in the **DataLookups to run on open** setting on the Control Sheet.

Parameter	Description
Order	Optional. Defines the processing order for the data source, among other data sources with the same name. Unnamed data sources are considered as having the same name for this purpose. Specify a whole number starting from 1. For example: `[DataLookup;FileGroup;2]` If the file contains multiple data sources with the name FileGroup, this data source will be executed second. If no order number is specified, then the data source is executed before any ordered data sources with the same name. Multiple data sources can be assigned the same order number. In this case, the rows of the data sources will be combined and executed in the same batch. When specific data source names are listed to execute, the overall order is determined by the order they are listed. For example, if **DataLookups to execute on open** is set to `FileGroup,Info`, then the FileGroup data source will be processed first. If there are multiple data sources named FileGroup, then those data sources will be processed in the order specified by the Order parameter, before moving on to process the Info data source. For more information, see Dependent data lookups.

NOTES:

- The primary DataLookup tag must be located in the first 500 rows of the sheet.
- The DataLookup tag can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Defining data lookup rows and columns

DataLookup data sources support the following row tags to query data. These tags must be placed in the control column, below the `[DataLookup]` tag.

Tag	Description
`[GetData]`	Returns data from a table, given a column name, a criteria statement, and a table name.

Tag	Description
[GetFeatureInfo]	Returns information about an installed product feature.
[GetFileGroupProperty]	Returns a specified property for a file group.
[GetFileGroupVariable]	Returns the value of a specified variable for a file group.
[GetFileGroupVariableEnablement]	Returns whether a picklist variable is enabled for a specified value in the picklist enablement column.
[GetFileGroupVariableProperty]	Returns information about a file group variable, especially picklist variable properties.
[GetPlanItemValue]	Returns values from the plan code table, for the current plan code (of a plan file).
[GetSecurityInfo]	Returns a specified security permission for a user.
[GetTableInfo]	Returns information about a specified table.
[GetUserInfo]	Returns a specified property for a user.

These row tags correspond to existing Axiom functions and return the same type of data. For example, if you would have used a GetData function to return the data, then you should use a `[GetData]` row tag within the DataLookup data source.

Each row tag has a set of query parameters that should be placed as column tags in the control row. For example, `[GetData]` rows use a parameter of `[ColumnName]`, to specify the database column from which to return data. For more information on the valid parameter columns for each row type, as well as data source examples, see the links in the previous table.

The parameter columns work as follows:

- All column tags must be placed to the right of the primary tag. Columns can be in any order within the control row.
- To omit a parameter for a particular query row, simply leave the cell blank. If a parameter is optional and you are not using it in any of the query rows, you can omit that column tag from the control row entirely.
- DataLookup data sources can contain different types of rows. The control row should contain the parameters necessary for all rows in the data source. Each row will only use the parameter columns that are relevant to that row; all other columns in the data source will be ignored for that row. If two separate row types use the same parameter name, then the same parameter column can be used for both types. For example, if the data source contains a `[GetFileGroupProperty]` row and a `[GetFileGroupVariable]` row, both rows can use the same `[FileGroup]` column.

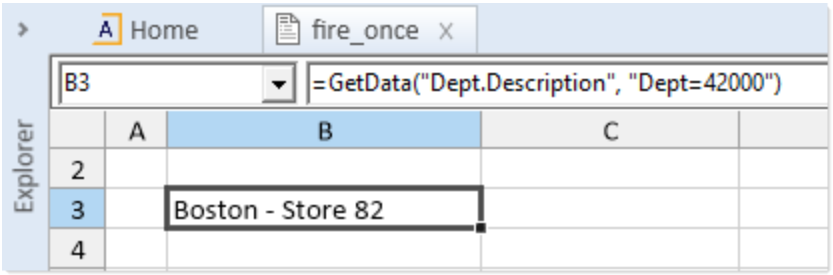
When the data lookup is executed, the result of each query row is always placed in the `[Result]` column for the data source. This is the one parameter column that is always required and is used by all query rows. Other areas of the file can reference the result cell to use the value returned for the query row.

Row and column tags do not need to be contiguous. Blank cells and cells with non-tagged text will be ignored in the control row and control column. Axiom will continue reading the control row and control column for tags until it encounters a tag that is not valid for use in the DataLookup data source, or until the end of the spreadsheet used range.

For information on using the Data Source Assistant to create or edit DataLookup data sources, see Using the Data Source Assistant to add or edit data lookups.

Example conversion of an Axiom function to a DataLookup data source

Imagine you have a GetData function as follows:



To return the same value using a DataLookup data source, you would create a data source with a `[GetData]` row, and complete the columns that correspond to the function parameters.

	A	B	C	D	E	F
6						
7		[DataLookup]	[result]	[iserror]	[columnname]	[filtercriteria]
8		[GetData]	Boston - Store 82	FALSE	Dept.Description	Dept=42000

When the data lookup is run, the result is placed in the `[Result]` column. If the data lookup returns an error, then the error text is placed in the `[Result]` column, and the `[IsError]` column returns True.

GetData data lookup

You can use GetData rows in a DataLookup data source to return data from a table, given a column name, a criteria statement, and a table name.

The GetData data lookup supports the same query parameters as the GetData function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetData data lookup, add a `[GetData]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetData parameters

`[GetData]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[ColumnName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
ColumnName	The name of the column to be queried. You can use an actual column name, a column alias, or a calculated field name. If the column name is fully qualified (Table.Column), then you can omit the TableName parameter. However, if you want to use multi-level column syntax (Table.Column.Column), then you must specify the TableName parameter. **NOTE:** If you are querying a system table—such as Axiom.Aliases—then you must enter only the column name here. Do not use fully qualified syntax with a system table. See the system table example below.
FilterCriteria	When querying data tables, the criteria statement specifies the records to be summed. If no criteria statement is specified, GetData returns the sum of the entire column. The criteria statement can be based on the data table, or on a lookup reference table that the data table links to (including multiple levels of lookup). When querying reference tables, the criteria statement typically specifies an individual record to be returned. The criteria statement can be based on the reference table, or on another reference table that it links to (including multiple levels of lookup). It is recommended to use fully qualified Table.Column syntax in the filter.

Parameter	Description
TableName	The name of the table to be queried. Note the following: • If the column name is an alias, then the table name can be omitted. Alternatively, you can specify "alias" as the table name, or specify the name of the table that the alias points to. • If the column name in the ColumnName parameter is fully qualified (Table.Column), then the TableName parameter can be omitted. However, if the ColumnName parameter uses multi-level column syntax (Table.Column.Column), then you must complete the TableName parameter.
NoDataDefaultMessage	Optional. A custom message to use as the return value if the query does not return any data. **NOTE:** Returning no data is not considered an error for purposes of the IsError column. IsError will return False when the no data message is displayed.
IgnoreSheetFilter	Optional. Specifies whether sheet filters are ignored for this query. • If FALSE, then sheet filters are applied to this query. False is the default value if this parameter is not specified. • If TRUE, then sheet filters are ignored for this query. All sheet filters are ignored, including temporary filters applied by Quick Filter and GetDocumentHyperlink, and multipass filters for file processing.
InvalidQueryMessage	Optional. A custom message to use as the return value if the database query is invalid.
AlternateAggregation	Optional. Specifies an alternate aggregation type for the returned data. In most cases this should be omitted to use the default aggregation for the column—for example, to sum data columns. The available aggregation types are the same as when using alternate aggregations with an Axiom query field definition. **NOTE:** When querying a system table (such as Axiom.Aliases), only Min and Max are supported. Other alternate aggregations are not supported and will return the same value as when using no alternate aggregation.

Parameter	Description
ConversionTarget	Optional. Specifies a conversion target so that the query returns converted data. This parameter only applies if the specified column and table have been configured for data conversions. You can specify a value directly (for example, "CAD"), or you can look up a value from a grouping column in a lookup reference table (for example, `DEPT.TargetCur`). In either case, the target must be present in the "to" column of the relevant conversion table. If the table is not configured for data conversions, an error results. If the specified conversion target is invalid, a zero value is returned. Data conversions will also return zero values if the conversion configuration for the table has invalid values, or if the conversion table itself has no valid rate entry for the relevant conversion and period (for example, blank entries in a rate column).
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

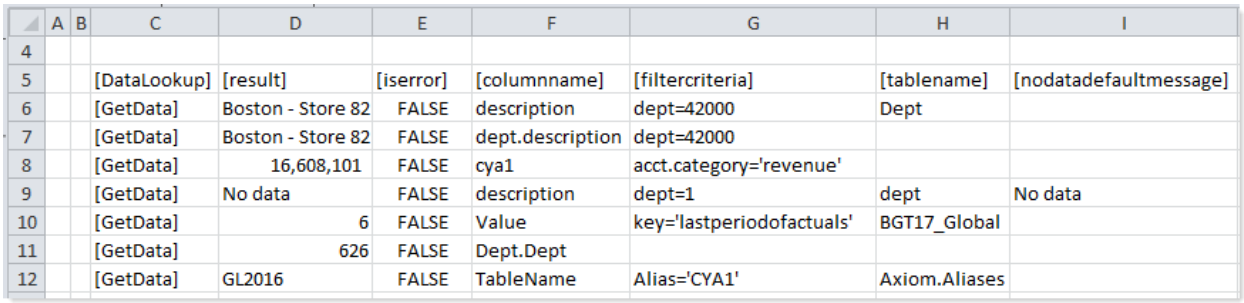
Remarks

- A user's table filters (as defined in Security) are always applied to GetData queries. Therefore, the value returned may vary depending on the user's table filters.
- If the column queried by GetData is a Boolean data type, then True values are returned as 1, and False values are returned as 0.

- If the column queried by GetData is a hyperlink column, then the raw value in the column is returned. The value is not auto-converted to a link (like when using Axiom queries). You can use the raw value returned by GetData in a variety of ways—for example, in a Hyperlink function, in a GetDocument function, in a Hyperlink component (forms), or in a HREF tag (forms).
- If you are using GetData queries in form-enabled files that will be open in a shared form instance (via use of embedded forms), then you may want to set the IgnoreSheetFilter parameter to True even if the sheet is not using sheet filters. This allows any duplicate GetData queries within the files to leverage the shared GetData cache. If IgnoreSheetFilter is False, then the GetData query is cached on a per sheet basis and the result cannot be shared with other sheets (regardless of whether the sheet actually has a defined filter).
- GetData can be used to query data from Axiom system tables, to return system information.

Example

The following screenshot of a DataLookup data source shows several GetData examples. Some optional parameter columns are omitted for space.



	A	B	C	D	E	F	G	H	I
4									
5			[DataLookup]	[result]	[iserror]	[columnname]	[filtercriteria]	[tablename]	[nodatadefaultmessage]
6			[GetData]	Boston - Store 82	FALSE	description	dept=42000	Dept	
7			[GetData]	Boston - Store 82	FALSE	dept.description	dept=42000		
8			[GetData]	16,608,101	FALSE	cya1	acct.category='revenue'		
9			[GetData]	No data	FALSE	description	dept=1	dept	No data
10			[GetData]	6	FALSE	Value	key='lastperiodofactuals'	BGT17_Global	
11			[GetData]	626	FALSE	Dept.Dept			
12			[GetData]	GL2016	FALSE	TableName	Alias='CYA1'	Axiom.Aliases	

For more examples of GetData use, see the GetData function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetFeatureInfo data lookup

You can use GetFeatureInfo rows in a DataLookup data source to return information about an installed product feature. This is typically only used by product developers.

The GetFeatureInfo data lookup supports similar query parameters as the GetFeatureInfo function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetFeatureInfo data lookup, add a `[GetFeatureInfo]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetFeatureInfo parameters

`[GetFeatureInfo]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
Code	Use one of the following keywords to specify what information you want to return about the feature: • `ID`: Returns the ID of the feature. • `Name`: Returns the name of the feature. • `Version`: Returns the version number of the feature. The version number is always zero-padded and includes at least three segments, such as: 01.02.00. • `DateApplied`: Returns the date the feature was last installed. All of these items return blank if the specified feature is not installed in the current system.
Feature	The name or ID of the feature for which you want to return information. The name must match the exact feature name as defined in the product package.

Example

The following screenshot of a DataLookup data source shows several GetFeatureInfo examples:

	A	B	C	D	E	F	G
4							
5			[DataLookup]	[result]	[iserror]	[featurename]	[codename]
6			[GetFeatureInfo]	3ca2f2da2eb54614b0	FALSE	Budget Planning	ID
7			[GetFeatureInfo]	Budget Planning	FALSE	Budget Planning	Name
8			[GetFeatureInfo]	2019.02.00.061	FALSE	Budget Planning	Version
9			[GetFeatureInfo]	6/14/2019 17:48	FALSE	Budget Planning	DateApplied

GetFileGroupProperty data lookup

You can use GetFileGroupProperty rows in a DataLookup data source to return information about a file group, such as the file group name or the current plan code.

The GetFileGroupProperty data lookup supports similar query parameters as the GetFileGroupProperty function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetFileGroupProperty data lookup, add a `[GetFileGroupProperty]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetFileGroupProperty parameters

`[GetFileGroupProperty]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
CodeName	Use one of the following codes to specify the information you want to return about the file group: • `Description`: Returns the file group description. • `DriversPath`: Returns the path to the Drivers folder for the file group. • `FileGroupID`: Returns the database ID of the file group. • `FileGroupName`: Returns the file group name. • `FileGroupDisplayName`: Returns the display name of the file group. • `FileGroupTabPrefix`: Returns the tab prefix of the file group. • `ProcessDefinitionID`: Returns the ID of the process definition associated with the file group. Only applies if a process definition is specified as the Plan File Process for the file group. Returns blank otherwise. • `PlanFile`: Returns the plan code for the current plan file. This keyword can only be used within a file group. When used in a file that is not a plan file (such as a template), the file name is returned. • `ScenarioName`: Returns the scenario name when the file group is a scenario (returns blank otherwise). • `UtilitiesPath`: Returns the path to the Utilities folder for the file group. • `TemplateName`: Returns the name of the template used to create the current plan file. Only applies to plan files.

Parameter	Description
FileGroup	Optional. The ID or name of the file group. • If you are using the data lookup within a file that belongs to a file group (template, driver, etc.), then you can omit this parameter and the current file group is assumed. • If you are using the data lookup outside of a file group (such as in a report file), then you must specify either the file group ID or the file group name (the true name, not the display name if it is different). File group aliases can be used for the file group name.
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

Example

The following screenshot of a DataLookup data source shows several GetFileGroupProperty examples:

	A	B	C	D	E	F	G
3							
4			[DataLookup]	[result]	[iserror]	[codename]	[filegroup]
5			[GetFileGroupProperty]	BGT17	FALSE	FILEGROUPTABPREFIX	
6			[GetFileGroupProperty]	25000	FALSE	PLANFILE	
7			[GetFileGroupProperty]	88	FALSE	FILEGROUPID	Budget 2017

For more examples of GetFileGroupProperty use, see the GetFileGroupProperty function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetFileGroupVariable data lookup

You can use GetFileGroupVariable rows in a DataLookup data source to return values for file group variables, given the variable names.

The GetFileGroupVariable data lookup supports similar query parameters as the GetFileGroupVariable function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetFileGroupVariable data lookup, add a `[GetFileGroupVariable]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetFileGroupVariable parameters

`[GetFileGroupVariable]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
VariableName	The name of the variable for which to return the value. You can return the value of any file group variable, including built-in variables and user-defined variables.
FileGroup	Optional. The ID or name of the file group. • If you are using the data lookup within a file that belongs to a file group (template, driver, etc.), then you can omit this parameter and the current file group is assumed. • If you are using the data lookup outside of a file group (such as in a report file), then you must specify either the file group ID or the file group name (the true name, not the display name if it is different). File group aliases can be used for the file group name.

Parameter	Description
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

▶ Example

The following screenshot of a DataLookup data source shows several GetFileGroupVariable examples:

	A	B	C	D	E	F	G
4							
5			[DataLookup]	[result]	[iserror]	[variablename]	[filegroup]
6			[GetFileGroupVariable]	2017	FALSE	FileGroupYear	
7			[GetFileGroupVariable]	2017	FALSE	FileGroupYear	Budget 2017
8			[GetFileGroupVariable]	2017	FALSE	FileGroupYear	88
9			[GetFileGroupVariable]	BGT2017	FALSE	PlanData	
10			[GetFileGroupVariable]	17	FALSE	ShortFileGroupYear	

For more examples of GetFileGroupVariable use, see the GetFileGroupVariable function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetFileGroupVariableEnablement data lookup

You can use GetFileGroupVariableEnablement in a DataLookup data source to return whether a picklist variable is enabled for a specified value in the picklist enablement column (True/False).

The GetFileGroupVariableEnablement data lookup supports similar query parameters as the GetFileGroupVariableEnablement function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetFileGroupVariableEnablement data lookup, add a `[GetFileGroupVariableEnablement]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetFileGroupVariableEnablement parameters

`[GetFileGroupVariableEnablement]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[VariableName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
VariableName	The name of the variable for which you want to return information.
EnablementValue	The enablement value to test. This is the integer code from the picklist that the picklist enablement column looks up to.
FileGroup	Optional. The ID or name of the file group. • If you are using the data lookup within a file that belongs to a file group (template, driver, etc.), then you can omit this parameter and the current file group is assumed. • If you are using the data lookup outside of a file group (such as in a report file), then you must specify either the file group ID or the file group name (the true name, not the display name if it is different). File group aliases can be used for the file group name.
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.

Parameter	Description
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

Remarks

- The picklist enablement column and the specified enablement values for each picklist variable are configured in the **Edit File Group** dialog, on the **Variables** tab, **Picklist Variables** sub-tab.
- GetFileGroupVariableEnablement returns True if the picklist variable is enabled for all values, or if the specified value is selected for the variable. It returns False if the picklist variable is disabled for all values, or if the specified value is not selected for the variable.
- If the specified variable is not a picklist variable, then GetFileGroupVariableEnablement always returns True. Enablement values do not apply to other variable types.
- If no picklist enablement column has been specified for the file group, then GetFileGroupVariableEnablement always returns True.
- GetFileGroupVariableEnablement does not evaluate whether the specified enablement value actually exists in the picklist that the picklist enablement column looks up to. An invalid enablement value will not cause GetFileGroupVariableEnablement to return an error.

Example

The following screenshot of a DataLookup data source shows several GetFileGroupVariableEnablement examples:

	A	B	C	D	E	F	G	H
2								
3			[DataLookup]	[result]	[iserror]	[variablename]	[enablementvalue]	[filegroup]
4			[GetFileGroupVariableEnablement]	TRUE	FALSE	Priority	4	21
5			[GetFileGroupVariableEnablement]	TRUE	FALSE	Reason	4	Capital Requests
6			[GetFileGroupVariableEnablement]	TRUE	FALSE	Priority	3	
7			[GetFileGroupVariableEnablement]	FALSE	FALSE	Reason	3	

For more examples of GetFileGroupVariableEnablement use, see the GetFileGroupVariableEnablement function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetFileGroupVariableProperty data lookup

You can use GetFileGroupVariableProperty in a DataLookup data source to return information about file group variable properties, given the name of a table variable or a picklist variable.

The GetFileGroupVariableProperty data lookup supports similar query parameters as the GetFileGroupVariableProperty function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetFileGroupVariableProperty data lookup, add a `[GetFileGroupVariableProperty]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetFileGroupVariableProperty parameters

`[GetFileGroupVariableProperty]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
VariableName	The name of the variable for which you want to return information.

Parameter	Description
CodeName	Use one of the following codes to specify the property you want to return for the variable: • `ColumnName`: Returns the name of the column on the plan code table that is associated with the variable. Returns blank if no column is associated with the variable. Applies to picklist variables. • `EnabledValues`: Returns a comma-separated list of enabled values for the variable. These values are selected for the variable based on the designated picklist enablement column in the plan code table. Values are returned using the integer code from the associated picklist, not the text value. Applies to picklist variables. Returns blank if the variable is enabled for all values, and returns the reserved text `Ax_Disabled` if the variable is disabled for all values. • `IsReadOnly`: Returns whether the variable is flagged as read-only (True/False). Applies to table variables. • `IsRequired`: Returns the variable is flagged as required (True/False). Applies to picklist variables.
FileGroup	Optional. The ID or name of the file group. • If you are using the data lookup within a file that belongs to a file group (template, driver, etc.), then you can omit this parameter and the current file group is assumed. • If you are using the data lookup outside of a file group (such as in a report file), then you must specify either the file group ID or the file group name (the true name, not the display name if it is different). File group aliases can be used for the file group name.
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.

Parameter	Description
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

Example

The following screenshot of a DataLookup data source shows several GetFileGroupVariableProperty examples:

	A	B	C	D	E	F	G	H
2								
3			[DataLookup]	[result]	[iserror]	[variablename]	[codename]	[filegroup]
4			[GetFileGroupVariableProperty]	Priority	FALSE	Priority	ColumnName	21
5			[GetFileGroupVariableProperty]	Reason	FALSE	Reason	ColumnName	Capital Requests
6			[GetFileGroupVariableProperty]	Ax_Disabled	FALSE	Priority	EnabledValues	
7			[GetFileGroupVariableProperty]	4	FALSE	Reason	EnabledValues	
8			[GetFileGroupVariableProperty]	FALSE	FALSE	Data	IsReadOnly	
9			[GetFileGroupVariableProperty]	FALSE	FALSE	Priority	IsRequired	

For more examples of GetFileGroupVariableProperty use, see the GetFileGroupVariableProperty function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetPlanItemValue data lookup

You can use GetPlanItemValue rows in a DataLookup data source to return values from the plan code table, for the current plan code. It is intended to be used in plan files.

When a plan file is opened, all values from the plan code table for that plan code are automatically loaded into document memory and can be retrieved using GetPlanItemValue. Using GetPlanItemValue instead of GetData eliminates the need to make round-trip server calls to the database to display this information in plan files.

The GetPlanItemValue data lookup supports similar query parameters as the GetPlanItemValue function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetPlanItemValue data lookup, add a `[GetPlanItemValue]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetPlanItemValue parameters

`[GetPlanItemValue]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[ColumnName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
ColumnName	The name of any column in the plan code table for the file group. Use only the column name, not full Table.Column syntax. For example, if you want to return a description from the Description column, use `Description` (*not* `Dept.Description`). GetPlanItemValue returns the value in this column, for the current plan code.

Remarks

- GetPlanItemValue only returns values when used in a plan file. When setting up the function in the file group template, it will return an expected error.
- The related values from the plan code table are loaded into document memory when the plan file is opened. If any of these values change while the plan file is still open in the current session, GetPlanItemValue will not reflect these changes. If a particular value may change often and you need to reflect this change in the current session, you should use GetData instead.

Example

The following screenshot of a DataLookup data source shows several GetPlanItemValue examples:

	C	D	E	F	G
5					
6		[DataLookup;Values]	[result]	[iserror]	[columnname]
7		[GetPlanItemValue]	25000	FALSE	Dept
8		[GetPlanItemValue]	Finance	FALSE	Description
9		[GetPlanItemValue]	Master Budget Template	FALSE	Template
10		[GetPlanItemValue]	Corporate	FALSE	WorldRegion

GetSecurityInfo data lookup

You can use GetSecurityInfo rows in a DataLookup data source to return information about security permissions for Axiom users, as defined in Security.

The GetSecurityInfo data lookup supports similar query parameters as the GetSecurityInfo function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetSecurityInfo data lookup, add a `[GetSecurityInfo]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetSecurityInfo parameters

`[GetSecurityInfo]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
CodeName	Specifies the security property to return. Use one of the following values: • `FileGroup`: Returns information on a user's plan file access filter. • `TableType`: Returns information on a user's table type access filter. • `Table`: Returns information on a user's table access filter. • `IsSysAdmin`: Returns whether a user is a system administrator. • `PermissionGranted`: Returns a user's security permission status. • `InRole`: Returns a user's role assignment status. For all codes except IsSysAdmin, you must use the SecurityItem parameter to specify which security item to return information about. For example, if you use the code FileGroup, you must use SecurityItem to specify which file group you want to return information about.
SecurityItem	The name of the security item for which to return information. This parameter is required for all code names except IsSysAdmin. The following list details which information to specify for each code: • FileGroup: Specify the name of the file group. A file group alias name can also be used. • TableType: Specify the name of the table type. • Table: Specify the name of the table. • PermissionGranted: Specify the code for the permission. See the following table for a list of permission codes. • InRole: Specify the name of the role.
UserName	Optional. The name of the user or role for which to return the security information. If omitted, the current user is assumed.
LocalizeResult	Optional. Only applies if the code name is FileGroup, TableType, or Table. Boolean value to determine whether the permission is returned using integer values or by using the permission name in security. • If FALSE (default), the function returns "0" to indicate no access, and "1" to indicate full access. • If TRUE, the function returns the permission names "No Access" or "Full Access". If the user's access is filtered, the filter text is returned.

Parameter	Description
ReturnWriteFilter	Optional. Only applies if the code name is TableType or Table. Boolean value to determine whether the user's read permissions or the user's write permissions are returned. • If FALSE (default), the user's read permissions are returned. • If TRUE, the user's write permissions are returned. By default, a user's read and write permissions are the same unless different write permissions are explicitly defined. So in most cases, returning the read filter returns the user's overall access. However, if you need to return the write access specifically, you can use this parameter to do so.
Domain	Optional. The Active Directory domain of the user. The domain property only applies to users that have been imported from Active Directory. Manually created users do not have a value for domain. You might be required to specify a domain to identify the correct user if users have been imported from multiple Active Directory domains and have the same user name. If all user names are unique across domains, then it is not necessary to specify the domain.
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

When using code PermissionGranted, the SecurityItem parameter must specify the name of the permission to return:

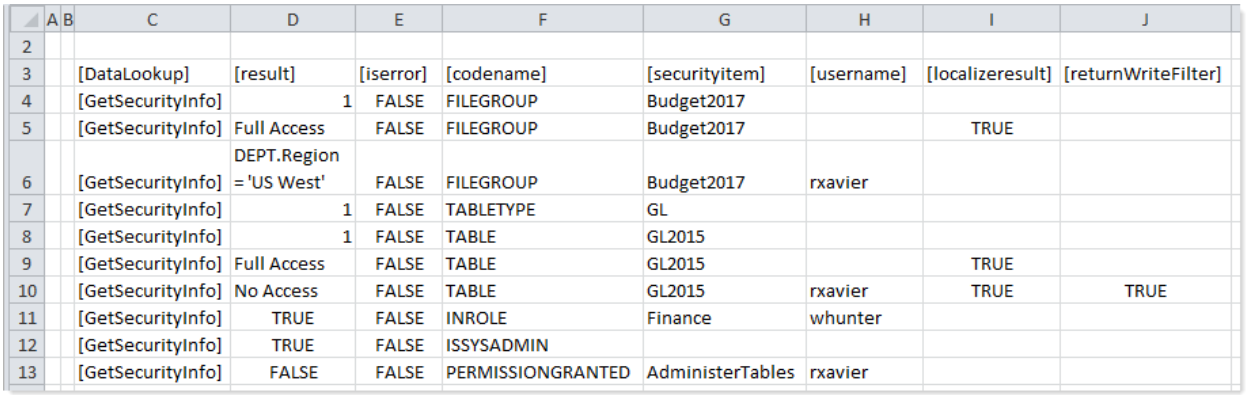
Security Permission	Value To Use In GetSecurityInfo
Administer Announcements	AdministerAnnouncements
Administer Axiom Explorer	AdministerRepository
Administer Ellucian Integration	AdministerEllucian
Administer Exports	AdministerExports
Administer File Groups	AdministerFileGroups
Administer Imports	AdministerImports
Administer Locked Items	AdministerLockedItems
Administer Picklists	AdministerPicklists
Administer Security	AdministerSecurity
Administer Tables	AdministerTables
Administer Task Panes	AdministerTaskPanes
Administer Updates	AdministerUpdates
Browse Audit History	BrowseAuditHistory
Remove Protection	RemoveWorkbookProtection
Scheduled Jobs User	ScheduledJobsUser
User Documents Folder Access	HasUserDocumentsFolder

Remarks

- The user's effective permissions are returned, including any access inheritance from a role or admin rights, and subsystem restrictions (if applicable).
- When using code FileGroup:
  - GetSecurityInfo only returns information on which plan files a user can access within the file group; it does not provide information on what level of access the user has to those files (for example, read only or read/write). Note that if a user has no access plus a filter, then GetSecurityInfo will return no access, even though the filter may be relevant if the user is eligible for permission "elevation" due to a plan file process.
  - Users can be granted different levels of access to different sets of plan files within the same file group. For the purposes of this function, all filters are concatenated with OR to result in the total set of plan files that the user can access at some level. If a subsystem restriction applies, it is concatenated with AND.
- When using codes Table, TableType, or FileGroup: if the filter uses a filter variable (such as `{CurrentUser.LoginName}`), the variable is resolved when GetSecurityInfo returns values for a user, and not resolved when GetSecurityInfo returns values for a role.

Example

The following screenshot of a DataLookup data source shows several GetSecurityInfo examples:



	A	B	C	D	E	F	G	H	I	J
2										
3			[DataLookup]	[result]	[iserror]	[codename]	[securityitem]	[username]	[localizeresult]	[returnWriteFilter]
4			[GetSecurityInfo]	1	FALSE	FILEGROUP	Budget2017			
5			[GetSecurityInfo]	Full Access	FALSE	FILEGROUP	Budget2017		TRUE	
6			[GetSecurityInfo]	DEPT.Region = 'US West'	FALSE	FILEGROUP	Budget2017	rxavier		
7			[GetSecurityInfo]	1	FALSE	TABLETYPE	GL			
8			[GetSecurityInfo]	1	FALSE	TABLE	GL2015			
9			[GetSecurityInfo]	Full Access	FALSE	TABLE	GL2015		TRUE	
10			[GetSecurityInfo]	No Access	FALSE	TABLE	GL2015	rxavier	TRUE	TRUE
11			[GetSecurityInfo]	TRUE	FALSE	INROLE	Finance	whunter		
12			[GetSecurityInfo]	TRUE	FALSE	ISSYSADMIN				
13			[GetSecurityInfo]	FALSE	FALSE	PERMISSIONGRANTED	AdministerTables	rxavier		

For more examples of GetSecurityInfo use, see the GetSecurityInfo function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

GetTableInfo data lookup

You can use GetTableInfo rows in a DataLookup data source to return information about a specified table.

The GetTableInfo data lookup supports similar query parameters as the GetTableInfo function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetTableInfo data lookup, add a `[GetTableInfo]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetTableInfo parameters

`[GetTableInfo]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
CodeName	Specifies the table property to return. Use one of the following values: • `Classification`: Returns the table classification for the table, such as Data or Reference. • `Exists`: Returns whether the table exists in the system (True/False). • `IndexScheme`: Returns the index scheme for the table, such as Default or Large Table. • `IsReadOnly`: Returns whether the table is read-only (True/False). • `TableType`: Returns the table type that the table is assigned to (blank if the table is not assigned to a table type). The following options only apply to product systems, and are configured by product development and implementation consultants: • `CustomerTableName`: Returns the real name of the table. • `IsUnused`: Returns whether the table is flagged as unused (True/False). Note that currently this flag does not affect the table; it is reserved for future functionality. • `IsVariable`: Returns whether the table is flagged as variable (True/False). • `PreferredName`: Returns the preferred name of the table.
TableName	The name of the table for which you want to return information. **NOTE:** If the table does not exist in the current system, the only code that will return a response is Exists. Otherwise, using an invalid table name will return an error.

Example

The following screenshot of a DataLookup data source shows several GetTableInfo examples:

	A	B	C	D	E	F	G
4							
5			[DataLookup]	[result]	[iserror]	[tablename]	[codename]
6			[GetTableInfo]		FALSE	Dept	TableType
7			[GetTableInfo]	Reference	FALSE	Dept	Classification
8			[GetTableInfo]	FALSE	FALSE	Dept	IsReadonly
9			[GetTableInfo]	Default	FALSE	Dept	IndexScheme
10			[GetTableInfo]	TRUE	FALSE	Dept	Exists

GetUserInfo data lookup

You can use GetUserInfo rows in a DataLookup data source to return information about Axiom users, as defined in Security.

The GetUserInfo data lookup supports similar query parameters as the GetUserInfo function, and can be used as a substitute for this function to improve file performance. The data lookup is intended to be used in cases where the queried value is not expected to change during the current file session, and therefore the value only needs to be queried once (or only after specific events).

To create a GetUserInfo data lookup, add a `[GetUserInfo]` row to a DataLookup data source and add the appropriate parameter columns. For more information on creating the DataLookup data source, see Creating DataLookup data sources.

GetUserInfo parameters

`[GetUserInfo]` rows use the following parameter columns. Within the DataLookup control row, these parameter names must be placed in square brackets—for example, `[CodeName]`. The parameters can be placed in any order.

NOTE: If the entry for any parameter depends on a value returned by another data lookup row, then that row must be in a different data source and executed before this row is executed. For more information, see Dependent data lookups.

Parameter	Description
CodeName	Specifies the user property to return. Use one of the following values: • `UserID`: Returns a user's database ID (a numeric value assigned by Axiom). • `UserName`: Returns a user's login name. • `UserEmail`: Returns a user's email address. • `FirstName`: Returns a user's first name. • `LastName`: Returns a user's last name. • `Domain`: Returns a user's Active Directory domain. This returns blank for users that were not imported from Active Directory.
UserName	Optional. The login name of the user for which you want to return information. If specified, the UserID parameter is ignored.
UserID	Optional. The database ID of the user for which you want to return information. This parameter can be used instead of UserName to specify a user.

Parameter	Description
Domain	Optional. The Active Directory domain of the user. The domain property only applies to users that have been imported from Active Directory. Manually created users do not have a value for domain. You might be required to specify a domain to identify the correct user if users have been imported from multiple Active Directory domains and have the same user name. If all user names are unique across domains, then it is not necessary to specify the domain.
Result	The Result column is where the return value for the row is placed when the data lookup is executed. You can reference this cell to use the return value in other areas of the file. This column must be present in order for the data lookup to be valid.
IsError	Optional. Indicates whether the return value for the data lookup was an error. • If TRUE, the return value is an error. This may be an Axiom #ERR code, a specific error message, or a custom error message defined in the data lookup (such as the GetData invalid query message). • If FALSE, the query executed successfully. The IsError column can be helpful if you need to set up formulas with error trapping. Instead of using the ISERROR Excel function, you can use a construction such as: `=IF(Info!$M$10=True, "", Info!$L$10)` Where the IsError column is in M10 and the Result column is in L10. If the data lookup returns an error, this function returns blank instead of displaying the error.

- If both UserName and UserID are left blank, then GetUserInfo returns information for the current user.
- If either UserName or UserID is specified, then GetUserInfo returns information for that user.

Example

The following screenshot of a DataLookup data source shows several GetUserInfo examples:

	A	B	C	D	E	F	G	H
3								
4			[DataLookup]	[result]	[iserror]	[codename]	[userid]	[username]
5			[GetUserInfo]	Wendy	FALSE	FIRSTNAME		
6			[GetUserInfo]	Jane	FALSE	FIRSTNAME		jdoe
7			[GetUserInfo]	Doe	FALSE	LASTNAME		jdoe
8			[GetUserInfo]	jdoe@fakeemail.com	FALSE	USEREMAIL		jdoe
9			[GetUserInfo]	2	FALSE	USERID		
10			[GetUserInfo]	whunter	FALSE	USERNAME	1	

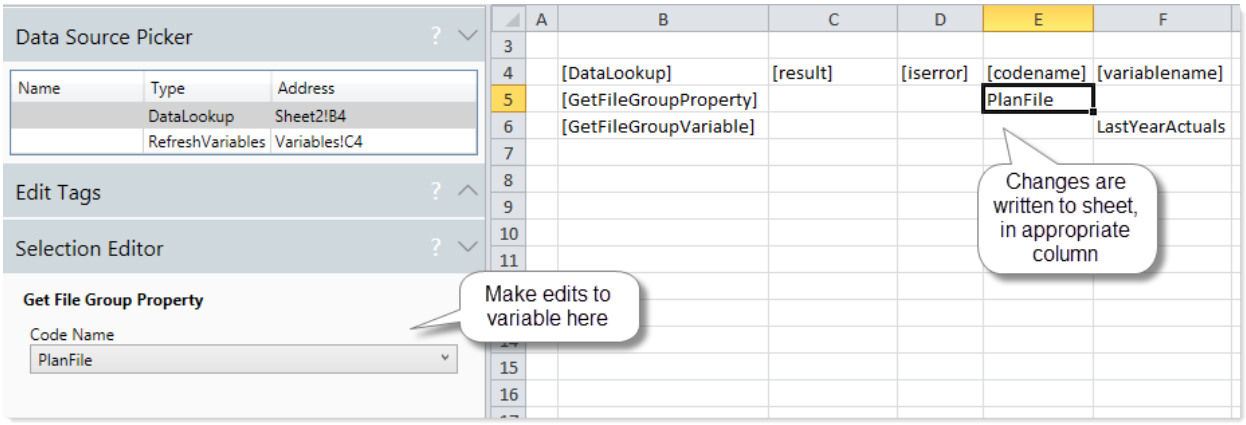
For more examples of GetUserInfo use, see the GetUserInfo function topic in the *Axiom Function Reference*. The same examples work for both approaches. To use a function example in a DataLookup data source, you would place the applicable function parameters in the corresponding parameter columns.

Using the Data Source Assistant to add or edit data lookups

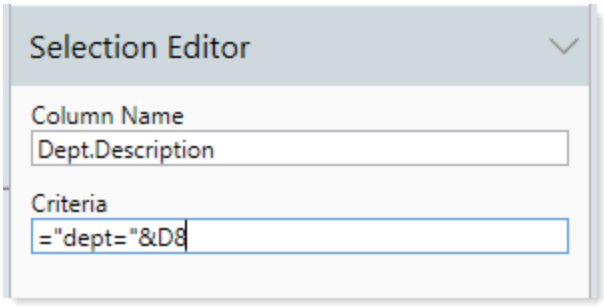
Once the DataLookup data source has been placed in a sheet, you can use the Data Source Assistant to add and edit the query rows, including adding optional parameter columns. For general information on the Data Source Assistant and its availability, see Using the Data Source Assistant.

Editing data lookup rows

To edit an existing data lookup row, place your cursor within the row, in the data source—either on the row tag or in one of the parameter columns. The parameters for the variable display in the **Selection Editor** section of the Data Source Assistant, filtered by row type. You can modify any of these parameters and the changes are written back to the sheet, into the appropriate columns.



If a particular parameter uses a formula, then that formula displays when you place your cursor in the property box. You can then edit that formula as needed.



The Selection Editor only displays parameters where the corresponding column already exists in the data source. For example, the parameter **File Group** only displays if the column tag `[FileGroup]` exists in the data source. If necessary, you can also use the Data Source Assistant to add missing tags to the data source. See Adding parameter columns.

Adding a new query row to the data source

To add a new row, use the **Edit Tags** section to add a row tag. You can do either of the following:

- Place your cursor in a blank row within the data source. Under **Row Tags**, click **Update** for the desired row tag. This adds the tag to the current row.
- Place your cursor in a populated row within the data source. Under **Row Tags**, click **Insert** for the desired row tag. This inserts a new row above the current row, and places the tag in the new row.

Once you have updated or inserted a row, you can go back to the Selection Editor section and define the parameters for that row.

NOTE: If the data source does not currently contain the required columns for the inserted row type, those columns will be automatically added to the end of the control row.

Adding parameter columns

When you use the right-click wizard to insert the DataLookup data source, only the minimum applicable column tags are placed in the sheet by default. If you want to use other optional tags, you must add them to the data source.

To add new column tags to a data source, use the **Edit Tags** section to do either of the following:

- Place your cursor in a blank column in the data source. Under **Column Tags**, locate the tag that you want to add and then click **Update**. This adds the tag to the current column.
- Place your cursor in a populated column within the data source. Under **Column Tags**, locate the tag that you want to add and then click **Insert**. This inserts a new column to the left of the current column, and places the tag in the column.

NOTE: The Edit Tags section only shows the column tags for the current row type. For example, if your cursor is in a `[GetData]` row, then only the columns that are valid for use with GetData rows are shown. Alternatively, you can place your cursor in the control row itself to add any column tag.

Once you have updated or inserted a new column tag, you can populate the column manually or you can use the Selection Editor section to update query rows for the new property. Remember that you can leave the column blank if it does not apply to a particular row.

Executing data lookups

Data lookups can be processed in the following ways:

- **On refresh**: If the DataLookup data source is unnamed, then it will be executed when the file is refreshed.
- **File open**: You can specify one or more DataLookup data sources to execute when the file is opened.
- **After Axiom query**: You can specify one or more DataLookup data sources to execute after a particular Axiom query is run.
- **On demand**: You can expose the Execute Data Lookups command in a ribbon tab, task pane, or Axiom form, so that users can execute one or more DataLookup data sources on demand.

Additionally, you may want to set up your file so that the data lookup results are automatically cleared when the file is saved. This option is typically used in report files to help ensure that users do not see data saved in the file by other users. To do this, use the setting **Clear DataLookups on save** in the **Workbook Options** section of the Control Sheet.

Executing on refresh

By default, all unnamed DataLookup data sources are executed whenever the file is refreshed. This includes refreshes that are triggered manually by a user (by clicking the **Refresh** button), as well as refreshes that happen automatically (such as refreshing an Axiom query on open or after saving data).

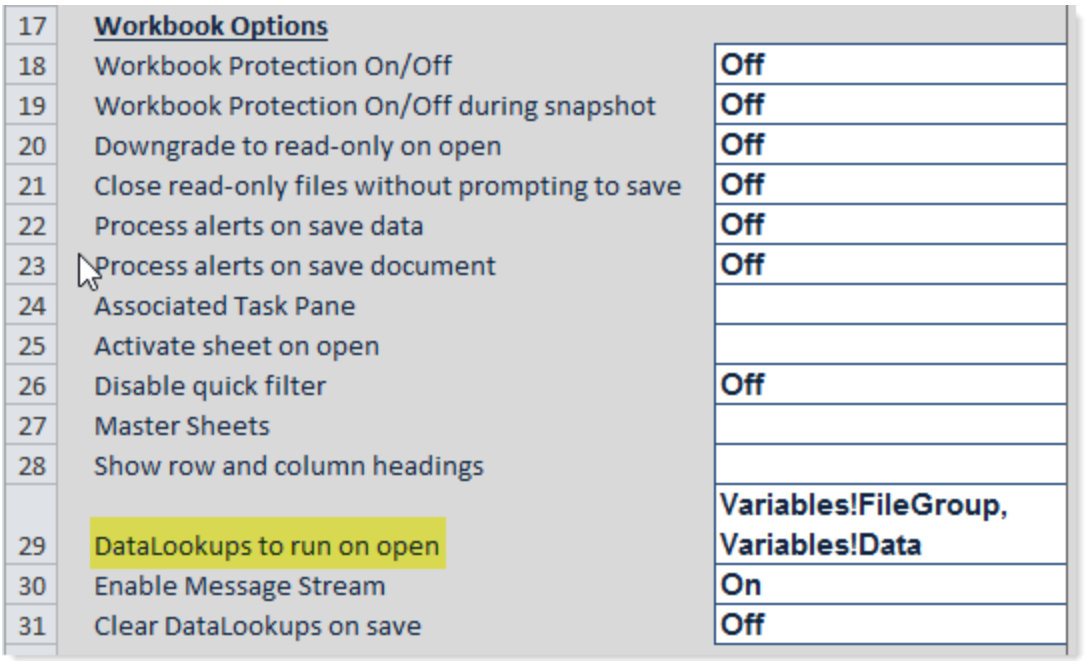
Therefore, you should only leave the data source unnamed if you want this automatic and potentially frequent refresh behavior. If the data lookup truly only needs to be executed once per file session, or only when certain other events occur, then you should give the data source a name and configure it to run only when necessary.

When data lookups are executed on refresh, they are executed after Axiom queries are run. Therefore, if the data source is built out using an Axiom query, it will be updated as part of the refresh.

Executing when the file is opened

You can specify one or more DataLookup data sources to execute automatically when the file is opened. This is the typical approach when the DataLookup data sources are used to retrieve "static" values that are then referenced throughout the file.

To do this, use the setting **DataLookups to run on open**, located in the **Workbook Options** section of the Control Sheet.



17	**Workbook Options**	
18	Workbook Protection On/Off	Off
19	Workbook Protection On/Off during snapshot	Off
20	Downgrade to read-only on open	Off
21	Close read-only files without prompting to save	Off
22	Process alerts on save data	Off
23	Process alerts on save document	Off
24	Associated Task Pane	
25	Activate sheet on open	
26	Disable quick filter	Off
27	Master Sheets	
28	Show row and column headings	
29	DataLookups to run on open	Variables!FileGroup, Variables!Data
30	Enable Message Stream	On
31	Clear DataLookups on save	Off

You can list multiple data sources in a comma-separated list, in the order you want them to be executed. Note the following:

- The list of data lookups can be qualified with sheet names, or unqualified. For example, if the data source is named Data, it can be listed as Sheet1!Data or just Data.
- If data source names are qualified with sheet names, then Axiom will only scan the listed sheets for the specified data sources. If the data source names are unqualified, then Axiom must scan all sheets for the data sources, which can impact performance. It is recommended to use qualified data source names whenever possible.
- The list of data lookups must be comprised of either all qualified names, or all unqualified names. Mixing qualified and unqualified names is not supported and will result in an error when the data lookups are executed.
- If you want to run unnamed data sources on open, you can use `SheetName![unnamed]` to run all unnamed data sources on the specified sheet only, or `[unnamed]` to run all unnamed data sources on all sheets. Remember that if you are also running an Axiom query on open, then the unnamed data sources will automatically be executed as part of that refresh.

Alternatively, you can configure a DataLookup data source to execute on open by giving it the reserved name **AxRefreshOnOpen**. Data sources with this name will automatically execute on open, after any other data sources listed in the Control Sheet property. You can have multiple data sources with this reserved name, and assign them an execution order in the DataLookup tag as needed.

When data lookups are executed on file open, they are executed after functions are calculated, and before "refresh on open" Axiom queries are run. Therefore, if you are building out the DataLookup data source via Axiom query, you should not use this setting. Instead, you should configure the Axiom query to refresh on open, and then execute the data lookup after the Axiom query.

If you are refreshing on open in conjunction with refresh variables, keep in mind the following:

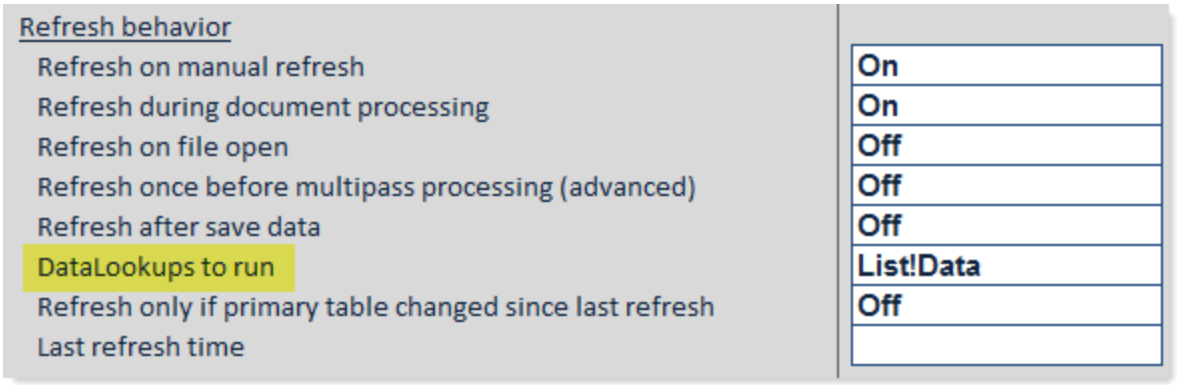
- The data lookups listed in **DataLookups to run on open** are run *before* refresh variables are processed. You should use this option if the refresh variable configuration depends on data returned by the data lookup.
- Data lookups that use the reserved name **AxRefreshOnOpen** are run *after* refresh variables are processed. You should use this option if the data lookup configuration depends on the user's refresh variable selections.

NOTE: When a file is opened using Open Without Refresh, the **DataLookups to run on open** setting is ignored. However, data lookups using the reserved name **AxRefreshOnOpen** are still run.

Executing after an Axiom query

For each Axiom query, you can specify one or more DataLookup data sources to execute automatically after the Axiom query is run. This option is intended to support executing named DataLookup data sources that are being constructed dynamically via Axiom query. The Axiom query builds out the data source rows, and then when the Axiom query is finished, the data lookups are executed.

To do this, use the setting **DataLookups to run**, located in the **Refresh Behavior** section of the Axiom query settings on the Control Sheet.



You can list multiple data sources in a comma-separated list, in the order you want them to be executed. Note the following:

- The list of data lookups can be qualified with sheet names, or unqualified. For example, if the data source is named Data, it can be listed as Sheet1!Data or just Data.
- If data source names are qualified with sheet names, then Axiom will only scan the listed sheets for the specified data sources. If the data source names are unqualified, then Axiom must scan all sheets for the data sources, which can impact performance. It is recommended to use qualified data source names whenever possible.
- The list of data lookups must be comprised of either all qualified names or all unqualified names. Mixing qualified and unqualified names is not supported and will result in an error when the data lookups are executed.

The `[unnamed]` keyword is not applicable here, because all unnamed data sources will be run anyway as part of the Axiom query refresh.

Executing on demand

Data lookups can be executed manually using the **Execute Data Lookups** command. You can list multiple data sources in a comma-separated list, in the order you want them to be executed. Note the following:

- The list of data lookups can be qualified with sheet names, or unqualified. For example, if the data source is named Data, it can be listed as Sheet1!Data or just Data.
- If data source names are qualified with sheet names, then Axiom will only scan the listed sheets for the specified data sources. If the data source names are unqualified, then Axiom must scan all sheets for the data sources, which can impact performance. It is recommended to use qualified data source names whenever possible.
- The list of data lookups must be comprised of either all qualified names or all unqualified names. Mixing qualified and unqualified names is not supported and will result in an error when the data lookups are executed.
- If you want to run unnamed data sources on open, you can use *`SheetName`*`![unnamed]` to run all unnamed data sources on the specified sheet only, or just `[unnamed]` to run all unnamed data sources on all sheets. Remember that if you are also running an Axiom query on open, then the unnamed data sources will automatically be executed as part of that refresh.

This command can be used to provide on-demand execution of data lookups. This command is not available to users by default—you must add it to a task pane, ribbon tab, or Axiom form before it is available for use. For more information on custom task panes and ribbon tabs, see the *System Administration Guide*. For more information on Axiom forms, see the *Axiom Forms and Dashboards Guide*.

Order of execution

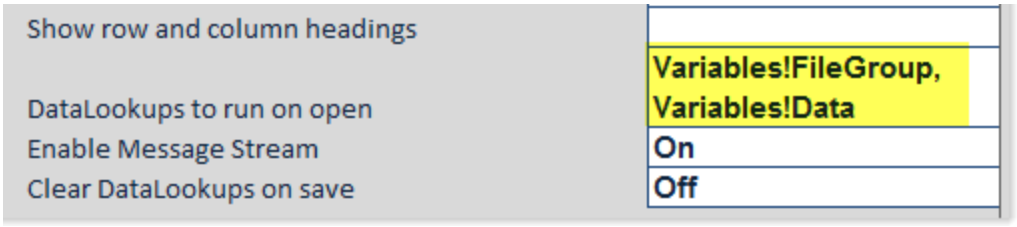
When data lookups are executed, they are executed in the order they are listed in the applicable setting. If multiple data sources have the same name, then the Order parameter in the `[DataLookups]` tag is used to determine the execution order of those data sources. Any unnamed data sources are treated as having the same name for this purpose. So if you need unnamed data sources to refresh in a particular order, they must use the Order parameter.

If one data lookup row is dependent on the result of another data lookup row, those rows must be in separate data sources and processed in the appropriate order. For more information, see Dependent data lookups.

Dependent data lookups

It is possible to configure data lookups so that one data lookup row uses the return value from another row in its query parameters. In order to do this, the data lookup rows must be in separate data sources, and the data sources must be processed in the appropriate order.

If both data sources have different names, then you can simply list them in the appropriate order when configuring the data lookup execution. For example, if a row in data source Data is dependent on a row in data source FileGroup, then you would list the data sources as follows: `FileGroup,Data` (either with sheet names or without).



This means the FileGroup data source will be executed first, followed by the Data data source. This dependent execution is necessary because the filter criteria statements for the GetData queries use the plan file code returned by the GetFileGroupProperty query.

	A	B	C	D	E	F	G
7							
8			[DataLookup;FileGroup]	[result]	[iserror]	[codename]	[variablename]
9			[GetFileGroupProperty]	4	FALSE	PLANFILE	
10			[GetFileGroupVariable]	CapReq	FALSE		CodeTable
11			[GetFileGroupVariable]	Capital2016	FALSE		PlanTable
12							
13							
14			[DataLookup;Data]	[result]	[iserror]	[columnname]	[filtercriteria]
15			[GetData]	New equipment	FALSE	CapReq.Description	CapReq=4
16			[GetData]	New equipment to support exp	FALSE	CapReq.LongDesc	CapReq=4
17			[GetData]	Type1	FALSE	CapReq.RequestType	CapReq=4
18			[GetData]	42000	FALSE	CapReq.Dept	CapReq=4

If both data sources have the same name, then you can use the Order parameter in the `[DataLookup]` tag to specify the order. This way you can list just one data source name to be executed, but execute both data sources in the appropriate order.

Show row and column headings	
DataLookups to run on open	**Variables!Info**
Enable Message Stream	**On**
Clear DataLookups on save	**Off**

For example, if both data sources are named Info, then you can specify the order of the first data source as 1 and the dependent data source as 2.

	A	B	C	D	E	F	G
7							
8			[DataLookup;Info;1]	[result]	[iserror]	[codename]	[variablename]
9			[GetFileGroupProperty]	4	FALSE	PLANFILE	
10			[GetFileGroupVariable]	CapReq	FALSE		CodeTable
11			[GetFileGroupVariable]	Capital2016	FALSE		PlanTable
12							
13							
14			[DataLookup;Info;2]	[result]	[iserror]	[columnname]	[filtercriteria]
15			[GetData]	New equipment	FALSE	CapReq.Description	CapReq=4
16			[GetData]	New equipment to support exp	FALSE	CapReq.LongDesc	CapReq=4
17			[GetData]	Type1	FALSE	CapReq.RequestType	CapReq=4
18			[GetData]	42000	FALSE	CapReq.Dept	CapReq=4

Dependent query rows must always be in separate data sources. It is not possible for a row in a data source to be dependent on a row in the same data source. All query rows that belong to the same data source are processed concurrently. The order of rows within the data source is irrelevant.

Dependency on Axiom functions

Data lookup parameters can reference the result of other Axiom functions. However, it is not recommended to reference the result of a GetData function. Instead you should use a separate DataLookup data source and set it up with a `[GetData]` row. You can then reference the result of that row, and set up the data lookup execution so that the data source with the GetData is executed first.

Generally speaking, if it is possible to return the value using a data lookup and you need to reference the value in another data lookup, then you should set those up as dependent DataLookup data sources rather than using a function and a data source.

6

Other Data Query Features

Axiom files can be configured so that certain actions occur as part of the process of updating the file with data. For example, you can present a refresh dialog to the user, prompting the user to select values that impact how the file is updated.

The features in this section apply to both Axiom queries and Axiom functions.

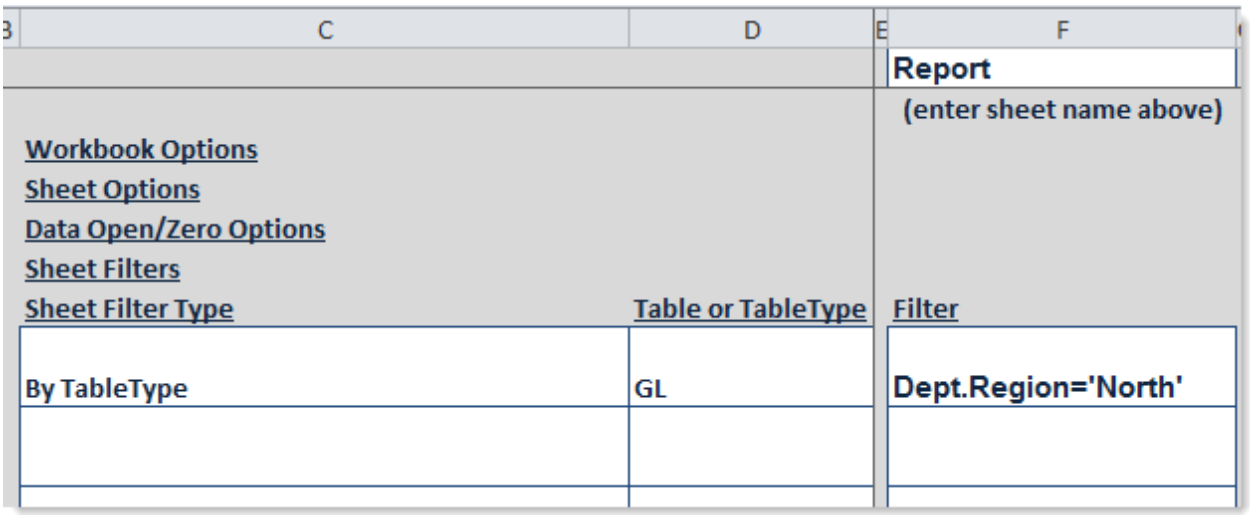
Defining sheet filters

On the Control Sheet, you can define a sheet-wide filter that filters the data to be returned by data queries on that sheet. The sheet filter applies to Axiom queries and GetData queries (function or data lookup).

If a sheet contains multiple queries and you want to use the same filter on all queries, then using the sheet filter may be easier and more efficient than duplicating the filter on each query.

Sheet filters can be defined by table or by table type. You may want to define a filter by table type if the primary table that is queried in the report is likely to change over time (but will belong to the same table type), or if the report queries data from multiple tables that belong to the same table type.

On the Control Sheet, the sheet filters are located in the **Sheet Filters** section:



	C	D	E	F
				Report
				(enter sheet name above)
	Workbook Options			
	Sheet Options			
	Data Open/Zero Options			
	Sheet Filters			
	Sheet Filter Type	Table or TableType		Filter
	By TableType	GL		Dept.Region='North'

In this example, one sheet filter type is defined, which filters the tables in the GL table type. The filter criteria statement `Dept.Region='North'` is applied to the Report sheet. Other sheets in this workbook could be filtered or not as desired.

NOTES:

- You can configure specific GetData queries to ignore the sheet filter if desired, using an optional parameter.
- If multiple filters are defined for the same sheet, those filters are concatenated using AND. Therefore, if you are filtering based on the same column, you should create a single compound filter rather than two separate filters. For example, if one filter is `Dept.Region='North'` and a second filter is `Dept.Region='South'`, those filters will be concatenated with AND, which will result in no data (because no department belongs to both the North region and the South region). Alternatively, a single compound filter such as `DEPT.Region='North' OR DEPT.Region='South'` (or `Dept.Region IN ('North', 'South')`) will return data.
- The Quick Filter feature and the function GetDocumentHyperlink can be used to apply a temporary sheet filter to a file. In this case, the temporary sheet filter does not display on the Control Sheet.
- Axiom system tables (Axiom.*TableName*) cannot be used in sheet filters.

To define a filter for a sheet:

1. In the **Sheet Filter Type** cell, use the drop-down list to select either **By Table** or **By Table Type**.
2. In the **Table or TableType** cell, type the name of the table or the table type (depending on the sheet filter type).

   If you do not know the exact name of the table or the table type, you can use the **Sheet Assistant** to look it up.

   NOTE: The entry here must represent the table or table type that you want to be filtered. For example, if you are querying tables from the GL table type and you want those tables to be filtered by a specific department, then the entry here should be the GL table type (*not* the Dept table).

3. In the **Filter** cell, in the definition column of the sheet to be filtered, enter the filter criteria statement.

   You can type the statement directly, or enter a cell reference to another cell that contains the statement, or right-click in the cell and select **Axiom Wizards > Filter Wizard**.

By default, the Control Sheet contains five filter rows. If the **Filter** box is blank for a sheet, then no filter will be applied to that sheet. You can define multiple filters to be applied to the same sheet, or to different sheets.

In most cases, the filter should be based on a lookup table. You can use full Table.Column syntax or column-only syntax, depending on the column to be filtered by. For example:

- `DEPT.Region='North'` limits the data to departments belonging to the North region.
- `DEPT>200` limits the data to departments greater than 200. Because the filter column is a validated key column, column-only syntax can be used to assume the lookup table (the filter is interpreted as `DEPT.DEPT>200`).

TIP: To validate that the sheet filter uses proper syntax, enable Developer Mode and then refresh the file. If the syntax is invalid, the error message will specifically identify the invalid sheet filter. Remember that if the sheet filter is built up using formulas or dependent on other data in the file, the filter may use valid syntax when the file is refreshed by one user but be invalid when refreshed by a different user. Files should be tested using a set of representative users to help catch these types of issues. For more information on developer mode, see Using Developer Mode.

Setting up refresh dialogs for Axiom files

You can configure an Axiom file so that a selection dialog displays when a refresh is initiated for the file. The user can select values in the dialog, and these values are then passed back to the file so that they can be used to impact the data refresh.

For example, you might want users to be able to filter a report on demand, so that the users only see the data for a selected department. You can use a refresh form to prompt the user to select a department when a refresh is initiated, rather than requiring the user to change values in the report itself. The selected department in the refresh form could then be used to filter an Axiom query in the file, or as a sheet filter.

Axiom provides two different options for defining refresh dialogs:

- **Refresh variables**: When using this approach, you define one or more refresh variables within the spreadsheet file. These variables represent the values that you want to collect from the user. When the file is refreshed, Axiom automatically generates a dialog that prompts the user to specify values for these variables. The variable values are then placed in designated cells within the file before the refresh occurs. The file must be configured to reference these values to impact the data query in some way.

  Refresh variables can be used in standard spreadsheet Axiom files, and in form-enabled Axiom files. For more information on this approach, see Refresh Variables.

- **Axiom forms**: When using this approach, you create a separate Axiom form to serve as the refresh form, and then configure the spreadsheet file to open this form as a dialog when a refresh occurs. The Axiom form must be designed to collect the desired values from the user, and to send those values back to the spreadsheet file using the "form state" feature. The spreadsheet file must be configured to reference these values to impact the data query in some way.

  The Axiom form option is only available for use with standard spreadsheet Axiom files. For more information on using this approach and creating the Axiom form, see the *Axiom Forms and Dashboards Guide*.

Generally speaking, the refresh variable approach is the simplest approach to set up. The refresh variables have a defined set of options that are specifically designed for this purpose. The refresh form is automatically generated by Axiom based on those options, so you do not have to separately design a dialog. All setup occurs within a single file, so it is easier for the file designer to understand and maintain.

The Axiom form approach requires more effort to set up, but it offers more visual flexibility than refresh variables. You can use the full range of Axiom form features to determine how the dialog should look and feel. However because this approach uses separate files, it is more complex to configure the data relationships between the files, and the files can potentially get out of sync.

These features are independent and should not be used together in a single workbook. If both features are present and both apply to the current refresh context, then the refresh variables will be used and the Axiom form will be ignored.

Querying data using data conversions

You can apply data conversion when querying a table, so that data returned from the query is converted according to the specified conversion target. For example, if a particular table stores data in U.S. dollars, you can convert the data as part of the query and return the values in Canadian dollars instead. This conversion happens on-the-fly whenever the query is refreshed; the converted data is not stored in the database.

The table that you are querying must already be configured to support data conversions. For more information on setting up data conversions for a table, see the *System Administration Guide*.

Data conversions can be performed when querying data via Axiom query, or when using GetData (either as a function or as a data lookup). When setting up an Axiom file to use data conversions, consider the following to help determine whether to use an Axiom query or GetData:

- When using GetData, you can configure the conversions on a cell by cell basis. Each individual GetData can be configured to perform data conversions or not, and can use different conversion targets. When using Axiom queries, each query can have only one conversion target, and all tables that are enabled for conversion will be converted. Depending on the desired report configuration, you may be able to use parallel Axiom queries to return a mix of converted and non-converted data, or to show different conversion targets side-by-side.

- Axiom queries allow you to override the default conversion type and/or the conversion scenario as part of the query. GetData functions do not support this override; a GetData function will always use the default conversion type and scenario.

The available conversion rates and conversion targets are defined by your organization. If you are not sure what conversions are available or which target codes to use, please contact your system administrator for assistance.

Setup considerations for using Quick Filter in a report

Using the Quick Filter feature, report users can easily apply a temporary filter to a report. Although this feature is available by default and does not require any specific setup, there are certain design considerations that report writers should keep in mind to improve the useability of the feature for end users.

There are three primary setup considerations regarding use of Quick Filter:

- **Enabling / Disabling Quick Filter.** By default, Quick Filter is enabled for all reports. If a particular report is not compatible with use of Quick Filter, then Quick Filter should be disabled for that report.
- **Defining hierarchies for use with Quick Filter.** The "Simple Filter" option for Quick Filter uses hierarchies defined in your system to provide a more user-friendly way to filter the data. You should make sure your system has useful hierarchies defined for use with this feature.
- **Displaying the Quick Filter in the report.** Once a Quick Filter has been applied, by default it is not visible in the report itself. Although users can go back to the Quick Filter dialog to view the current filter, the "best practice" approach is to display this filter in the report using the GetCurrentValue function.

One additional consideration is the behavior of the Quick Filter dialog when using Axiom queries versus GetData functions in the report. When using Axiom queries, the dialog is filtered to only show the hierarchies and tables that are relevant to the Axiom queries in the report (based on the primary tables). When using GetData functions, this filtering is not possible and all available hierarchies and reference tables are shown. It is good to be aware of this difference in behavior, as it may influence the report design or determine whether or not you disable Quick Filter for the report.

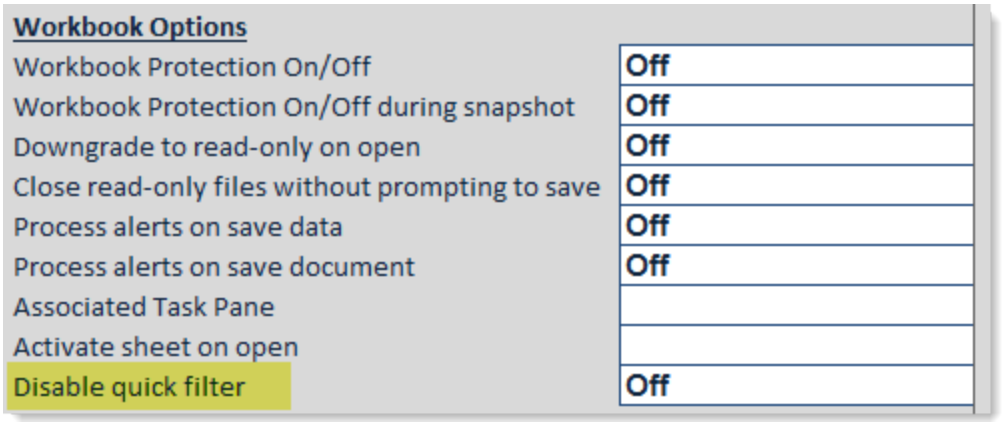
NOTE: The Quick Filter feature can also be used on file group utilities and drivers. If you intend for users to use Quick Filter in these files, the same setup considerations apply.

▶ Enabling or disabling Quick Filter for a report

By default, the Quick Filter feature is available for all reports in your system. The feature is not controlled by security; it is available to any user who can open a report.

However, you may have certain reports that are not compatible with use of Quick Filter. For example, the report may be designed so that it brings in a particular set of data for a particular purpose, and you do not want end users to be able to apply a Quick Filter and see potentially confusing results. In this case, you can disable Quick Filter on a per file basis.

The option that controls the availability of Quick Filter is named **Disable quick filter**, and it is located in the **Workbook Options** section of the Control Sheet. By default, this option is set to **Off** so that Quick Filter is available, but you can change it to **On** to disable Quick Filter. If Quick Filter is disabled for a file, then by default the **Quick Filter** button on the Axiom ribbon tab will be grayed out and unavailable for use.



Workbook Options	
Workbook Protection On/Off	Off
Workbook Protection On/Off during snapshot	Off
Downgrade to read-only on open	Off
Close read-only files without prompting to save	Off
Process alerts on save data	Off
Process alerts on save document	Off
Associated Task Pane	
Activate sheet on open	
Disable quick filter	Off

The **Disable quick filter** option is applied when the file is opened. If you change this option in a file, you must close and reopen the file to see the effect on the Axiom ribbon tab (or on any custom ribbon tabs or task panes where you have used the Quick Filter command).

If you have a very controlled system and you do not want end users to ever have access to the Quick Filter feature, then you can create a custom Axiom ribbon tab for those users and remove the Quick Filter command from that ribbon tab. For more information, see the *System Administration Guide*.

Defining hierarchies for use with Quick Filter

The Quick Filter feature supports two options for defining the filter:

- **Simple Filter** uses the hierarchies defined in your system to provide a more user-friendly way of defining filters. As a user selects items in the hierarchy, Axiom builds the necessary filter syntax for the user.
- **Advanced Filter** allows the user to craft a filter based on any valid table column, however, the user must have at least a basic understanding of filter syntax and your table structure to use this option.

If no hierarchies are defined in your system, then the only option available to users is the Advanced Filter option. If your end users will be using Quick Filter, you should make sure that your system has useful hierarchies for this purpose.

Hierarchies are defined on reference tables, and are used throughout the system to help define filters and drilling paths. For more information on setting up hierarchies for your system, see the *System Administration Guide*.

Once hierarchies have been defined, it is recommended to test Quick Filter and drill-down drilling on some representative files, to make sure that the hierarchies provide the desired options to end users.

Displaying the current Quick Filter in the report

When a user applies a Quick Filter to a report, that filter is not visible in the report file itself. By default, the only way that a user can see the current Quick Filter is to reopen the Quick Filter dialog and look at the filter listed there.

If you anticipate that users will use Quick Filter often on a particular report, that report should be set up with a GetCurrentValue("QuickFilter") function in the report header. This function will display the current Quick Filter applied to a report. This makes the filter more obvious to the user and makes it less likely that the user will forget that a filter has been applied to the report. For more information on using this function, see the *Axiom Function Reference*.

Refresh Variables

Refresh variables can be used to prompt the user to specify certain values before refreshing the file. You can configure the file so that these values impact the data refresh in some way, such as to filter the data coming into an Axiom query.

The refresh variables feature is an important feature for both file designers and end users. Refresh variables provide an easy-to-use and standardized way to allow users to filter the data coming into a file.

Refresh variables are defined on a sheet in the file, using a RefreshVariables data source. Each row in the data source defines a variable representing a value that you want the user to input or select. Several different variable types are available to handle different kinds of values, from free-input text or numbers, to selecting values from defined lists or designated table columns.

Refresh variables can be used in two different environments:

- In standard Axiom files where users are viewing the file as a spreadsheet in the Desktop Client (Excel Client or Windows Client). The refresh variables can be triggered to display when the file is initially opened, and/or when the user manually refreshes the file.
- In form-enabled files where users are viewing the file as an Axiom form. The refresh variables display when the user clicks the Filters button in the Web Client Task Bar. If the Axiom form is open within the Desktop Client, then the refresh variables are available in the Filters task pane.

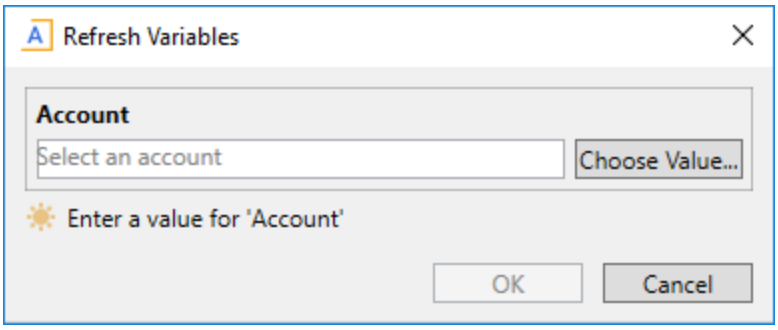
How refresh variables work

An Axiom file can have any number of refresh variables. Refresh variables are defined by creating a RefreshVariables data source on any non-control sheet in the file. This data source is a series of tags that defines one or more refresh variables and their properties.

Refresh variable behavior for spreadsheet Axiom files

When a user refreshes a spreadsheet Axiom file in the Desktop Client (Excel Client or Windows Client), Axiom checks for the presence of a RefreshVariables data source. If one is found, then the refresh variables in the data source are presented to the user in an automatically generated dialog.

Within the refresh dialog, the variables are displayed in the order that they are listed in the data source. Once the user specifies values for the variables, these values are placed in the `[SelectedValue]` column of the data source, and the refresh proceeds.



Example refresh dialog generated from a refresh variable

NOTES:

- If a user cancels out of the refresh dialog, then the data refresh does not occur.
- Refresh variables apply to the entire workbook. If the refresh is for the current sheet only, refresh variables are still presented to the user, even if they are defined on a different sheet.
- By default, refresh variables are ignored when the file is refreshed on open. If you want the refresh dialog to display on open, then you can change the Refresh Form Run Behavior for the file, as defined on the Control Sheet. For more information, see Determining when refresh variables display to users.

In order for the refresh to be impacted by the user's choices, the file must be set up to use these values in some way—such as in an Axiom query filter or a sheet filter. This part is entirely up to the file designer; Axiom does not do anything with the user's selected values other than place them in the appropriate cells within the file.

Refresh variables are ignored and the refresh dialog does not display when the refresh is performed as part of an automated process, such as:

- Refreshes that occur as part of File Processing
- Refreshes performed by the **Process Plan Files** utility
- Refreshes that occur when a file is processed by a Scheduler task

Refresh variable behavior for Axiom forms

When users view Axiom forms, refresh variables display within the Filters panel. Users can select values for the variables within the panel, and then apply the values to the form. The Web Client Container must be enabled for the form in order to access the Filters panel.

For more information on how refresh variables work with Axiom forms, see the *Axiom Forms and Dashboards Guide*.

File designers should have a good understanding of how Axiom forms work before attempting to set up refresh variables for use in a form-enabled file. Although the setup of the refresh variables is essentially the same, the overall design of the file and how it presents data to users is different than when using standard spreadsheet Axiom files.

NOTE: Refresh variables can also be used with web reports created in the Report Designer. However, the available features and the setup is very different than when using refresh variables in other files. For more information, see the *Web Reports Guide*.

Defining refresh variables

Refresh variables are defined by using a RefreshVariables data source. This data source is a series of tags that define the refresh variable properties.

Each Axiom file can have a single RefreshVariables data source that is placed on any non-control sheet in the file. In practice, this data source is typically placed on a designated sheet named something like Variables. If the file is a standard Axiom spreadsheet file that users will access in the Excel Client or Windows Client, then this sheet is typically hidden from end users (though it does not have to be).

To create a RefreshVariables data source:

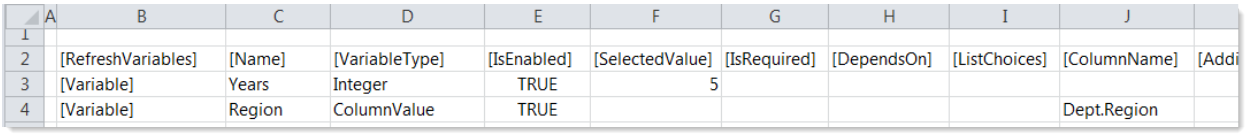
- Right-click the cell in which you want to start the data source, then select **Axiom Wizards > Insert Refresh Variable Data Source**.

The wizard adds the primary tag, all column tags, and one row tag to define a single variable. To create variables, you can manually populate the data source, or you can use the Data Source Assistant. For more information, see Using the Data Source Assistant to add or edit refresh variables.

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 screenshot shows an example RefreshVariables data source with two variables:



	A	B	C	D	E	F	G	H	I	J	
1											
2		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[DependsOn]	[ListChoices]	[ColumnName]	[Addi
3		[Variable]	Years	Integer	TRUE	5					
4		[Variable]	Region	ColumnValue	TRUE					Dept.Region	

The RefreshVariables data source uses the following syntax:

Primary tag

[RefreshVariables]

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

[Variable]

Each row flagged with this tag defines a unique refresh variable. Generally speaking, each variable represents a value that you want the user to input or select.

Column tags

Each column in the data source defines a variable property, such as the variable name and type, and whether the variable is enabled. All variables share a set of general properties. Additionally, certain variables have additional properties that only apply to that particular variable type.

[VariableType]

This column defines the variable type, which determines what users can enter for the variable. The variable type also determines the valid property columns for the variable.

For more information on completing the property columns for a variable type, see the individual topics on each type (listed below).

- **AdvancedFilter**: The user can create a filter criteria statement using the Advanced Filter view, or create a limit query statement.
- **Calendar**: The user can select a date from a calendar.
- **CheckBox**: The user can select or clear a check box.
- **ComboBox**: The user can select a value from a drop-down list. The list can be generated based on a specified table column, or an Axiom query, or a ComboBox data source. The user can type into the box to filter the items in the list.
- **Decimal**: The user can enter any numeric value, including decimals.
- **Grid**: The user can select a value from a specified table column (for example, ACCT.ACCT to select from a list of accounts). The column values are presented in a searchable grid dialog.
- **HierarchyFilter**: The user can select one or more values from a defined hierarchy to result in a filter criteria statement.
- **Integer**: The user can enter any whole number.

- **RadioButton**: The user can select a single value from among two or more radio buttons. The list of radio buttons can be generated based on a specified table column, or an Axiom query, or a ComboBox data source.
- **RangeSlider**: The user can select the top and bottom values within a defined range, using slider buttons.
- **RelatedColumnValue**: This is a special variable type that is not presented to the user. Instead, it is used to return a related column value for a parent variable, so that the value can be referenced in report titles or other settings.
- **Slider**: The user can select a single value within a defined range, using a slider button.
- **String**: The user can enter any string value.
- **StringList**: The user can select any item in a manually defined list.

Note the following about placement of the data source tags:

- Column tags can be in any order. Optional column tags can be omitted from the data source if they are not being used. For example, if none of your variables are StringList type, you can omit the `[ListChoices]` tag.
- Column and row tags do not have to be continuous. Axiom will continue searching the control row and control column for valid tags until it reaches another tag of any type.

Testing refresh variables

To validate the refresh variable properties, perform a refresh in the Desktop Client. If any invalid settings are present, an error message will display. You should do this even if the intended use of the file is the Web Client, to catch any invalid settings.

Once you have set up the file as desired and corrected any invalid settings, you should test the refresh variables in their intended environment (Desktop Client refresh dialog or Web Client filter panel). Make sure that:

- The variable name and selections make sense from a user perspective. If it does not seem clear what you are asking the user to do, you may want to edit the variable name or use a different variable type.
- If the variable is not required, test a blank entry to make sure that the report still makes sense if no value is specified. If not, you may want to edit the report design or make the variable required.
- If you are using dependent variables, make sure that any dynamic settings are working as expected. Try entering different values for the parent variable to make sure you have accounted for all possibilities.
- Lastly, test to make sure the data refresh is affected by the refresh variables as you intended. Remember that it is up to the file designer to reference the user's selected values as necessary to impact the data refresh.

Once you have finished testing the file, remember to clear the contents of the [SelectedValue] column before saving the file (or restore these values to your intended "default" values). You can also use the [ClearSelectedValueOnSave] column or the [ClearSelectedValueOnOpen] column to automatically clear values for you.

General refresh variable properties

The following variable properties apply to all refresh variable types.

Column Tag	Description
[Name]	The name of the variable. This name identifies the variable row in the data source, and is also used as the variable display name to users if a separate display name is not defined in the [DisplayName] column. The name should not contain any non-alphanumeric characters such as question marks or periods. If you want the variable name that displays to users to include non-alphanumeric characters, use the display name. The name cannot be dynamic; it must remain static because it is used to identify the variable. If you are configuring a dependent variable and you need the name to change based on the selection of the parent variable, then you must make the display name dynamic instead of the name.
[DisplayName]	Optional. The display name of the variable. If defined, the display name will be used instead of the name when the variable displays to users.
[VariableType]	Specifies the type of variable, such as String, Grid, or ComboBox. The variable type determines the presentation of the user input, as well as the valid values. Most variable types use additional property columns to define the contents and behavior of the variable. For more information, see the topics on each specific type.

Column Tag	Description
[IsEnabled]	Specifies whether the variable is enabled (True/False). • If True, then the variable will be included in the component. • If blank or False, then the variable will not be included in the component. This evaluation is determined when the component is opened. If the variable is flagged as dependent using the [DependsOn] column, then the evaluation will be performed again after a value has been selected for the parent variable.
[SelectedValue]	The user's entered value for the variable will be placed in this cell. When setting up the file to use the variable value, point your formulas to this cell. If you want to define a default value for the variable, use the [DefaultValueonOpen] field rather than entering the default value here.
[ClearSelectedValueonSave]	Optional. Specifies whether the selected value is cleared when the file is saved (True/False). • If True, then the selected value is cleared when the file is saved. You should set this to True if you want to ensure that the variable always starts off with no value after saving. • If blank or False, then the selected value is left as is when the file is saved. You should set this to False if you want to be able to set a "default value" for the variable, or if you want the last-selected value to be retained after saving the file.

Column Tag	Description
[ClearSelectedValueonOpen]	Optional. Specifies whether the selected value is cleared when the file is opened (True/False). • If True, then the selected value is cleared when the file is opened. You should set this to True if you want the variable to start with no value when the file is opened. • If blank or False, then the selected value is not cleared when the file is opened. You should set this to False if you want the last-saved value to be retained when opening the file. If you enable this option and also define a default value using [DefaultValueonOpen], then the current selected value is first cleared, then the default value is copied to the [SelectedValue] field.
[IsRequired]	Optional. Specifies whether the user must enter a value for this variable (True/False). • If True, then the user must specify a value for this variable in order to perform the refresh. • If blank or False, then the user can leave the variable blank. The file should be configured so that it works as expected if the variable is left blank. The display of required and optional variables depends on the environment. In the Desktop Client, the text **(optional)** follows the name of optional variables. In the Web Client, required variables that do not yet have a selected value are indicated with a red bar along the side of the variable field.
[DependsOn]	Optional. Specifies that the variable is dependent on a "parent" variable. To make a variable dependent on another variable, enter the name of the parent variable. Some variable types require a parent variable, such as RelatedColumnValue variables. Other variable types can be made dependent as needed. Dependent variables can be updated dynamically in response to the selected value for the parent variable. For more information, see Using dependent refresh variables.

AdvancedFilter refresh variable

AdvancedFilter refresh variables prompt users to create a filter criteria statement using the advanced view of the Filter Wizard. This statement can optionally be applied as a Quick Filter during the refresh, or the statement can simply be returned into the `[SelectedValue]` column for the file designer to use as needed.

When using the AdvancedFilter refresh variable, you must specify a primary table to set the context for the Filter Wizard. The Filter Wizard is then limited to only creating filters that are valid for use against that primary table.

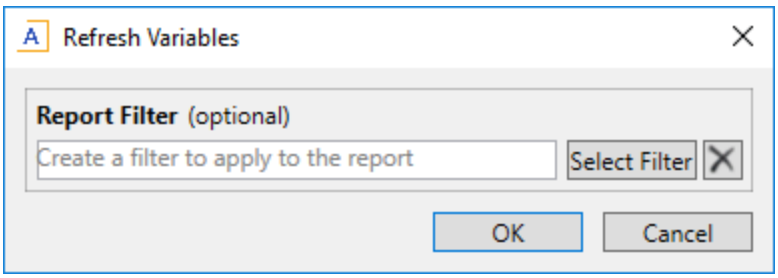
For example, you can prompt users to create a filter for the GL2020 primary table. The user might create a filter such as `Dept.WorldRegion='Europe'` (because Dept is a lookup table for GL2020). The file designer can set up the file so that this filter is referenced by an Axiom query data filter, or the file designer can configure the variable so that the statement is automatically applied as a Quick Filter.

The AdvancedFilter refresh variable can also be used to create a limit query statement, which can be used in Axiom queries to limit the query based on data in another table. In this case the return value is not simply a filter criteria statement, but full limit query syntax that can be referenced by the **Limit query data based on another table** setting for Axiom queries.

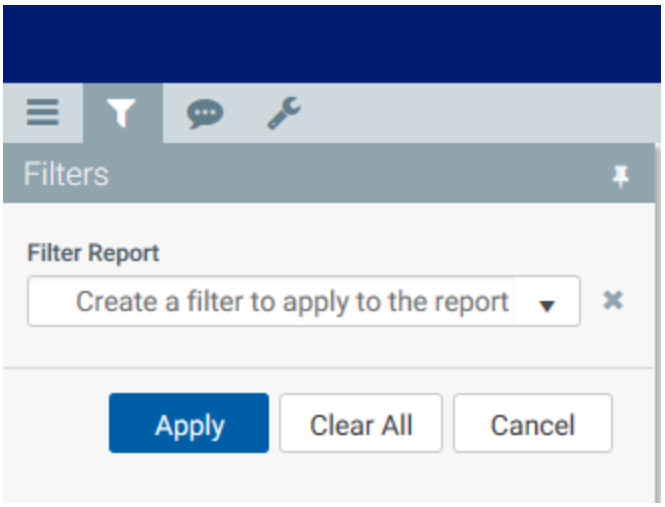
NOTE: The AdvancedFilter refresh variable should only be used when the target audience for the file is advanced users who understand your system's data structures and filter criteria syntax.

Variable behavior

When using the variable to create a standard filter criteria statement, the variable displays as a read-only text box with a button next to it. The user can click the button to create the filter.

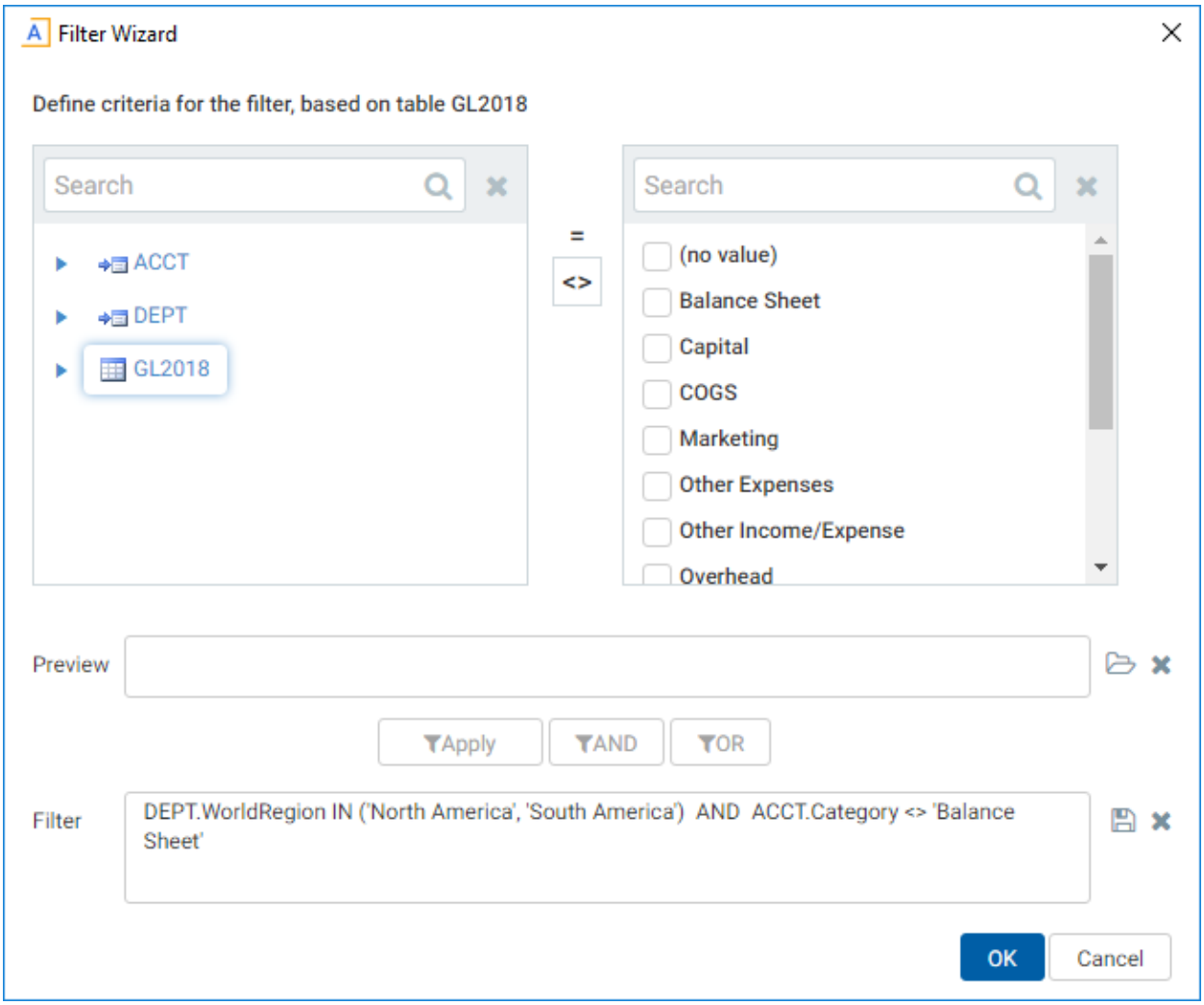


Desktop Client: Example AdvancedFilter refresh variable



Web Client: Example AdvancedFilter refresh variable

Clicking the button opens the **Filter Wizard** dialog in advanced view. The user can create a filter by selecting a table column, an operator, and a value. The dialog for the filter creation is the same in all clients.



Example Filter Wizard dialog

Once the user makes selections, a filter criteria statement is placed in the **Preview** box. The user must click **Apply** to move the preview filter down to the **Filter** box. If the user wants to make a compound criteria statement, they can make more selections and then use **AND** or **OR** to combine the new statement with the existing statement. The user can make manual edits to the filter at any point.

When the user clicks **OK**, the statement in the **Filter** box is written back to the `[SelectedValue]` field for the variable, where it can be referenced by Axiom queries or other queries as needed. Or if `[UseAsQuickFilter]` is enabled, the filter is automatically applied as a Quick Filter, with no formulas needed.

See Variable-specific properties (limit query creation) for more information on the user experience when the variable is used to create a limit query statement.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining an AdvancedFilter variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define AdvancedFilter variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties (standard filter creation)

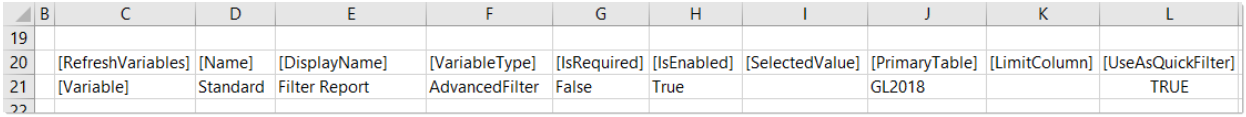
The following additional properties apply to AdvancedFilter variable types, when using the variable to create a standard filter criteria statement:

Column Tag	Description
`[PrimaryTable]`	The primary table to determine the table choices available in the Filter Wizard. Basically, the primary table is the table you want to filter against, so you want the Filter Wizard to display tables that are valid for filtering the primary table. The Filter Wizard displays the primary table itself, as well as any reference tables that the primary table looks up to. • If the primary table is a data table, the lookup reference tables are listed underneath the primary table and also separately. This allows creating a filter criteria statement that starts from the primary table or one that starts from a lookup reference table. For example, the end filter can be either `Dept.WorldRegion='Europe'` or `GL2020.Dept.WorldRegion='Europe'`. • If the primary table is a reference table, the lookup reference tables are only listed underneath the primary table. In this case, the filter criteria statement is only valid if it starts from the primary table.

Column Tag	Description
[UseAsQuickFilter]	Optional. Specifies whether the filter criteria statement created in the Filter Wizard is applied as a Quick Filter. • If True, then the filter criteria statement is applied as a Quick Filter to the workbook. This works just like when using the Quick Filter dialog to apply a temporary filter to the workbook. If a Quick Filter is already applied to the workbook, the old filter is cleared and the new filter replaces it. Only one AdvancedFilter variable can be applied as a Quick Filter per file. **NOTE:** The Quick Filter is applied to the entire workbook. The option to apply a Quick Filter to only the active sheet is not available when applying a Quick Filter via refresh variables. • If blank or False, then the filter criteria statement is not applied as a Quick Filter. If you want to filter data based on this statement, then you must configure the file to do so, such as referencing the statement in the data filter for an Axiom query.
[PlaceholderText]	Optional. Defines placeholder text to display within the variable box until a value is selected. This text also displays as a tooltip for the **Select Filter** button.

The following properties do not apply to AdvancedFilter variables when used to create a regular filter criteria statement: ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, Hierarchies, DataSourceName, DisplayFormat, MinDate, MaxDate, TooltipColumn, AutoQuoteString, LimitColumn, MinValue, MaxValue, StepFrequency.

The following screenshot shows an example AdvancedFilter variable to create a regular filter criteria statement:



	B	C	D	E	F	G	H	I	J	K	L
19											
20		[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsRequired]	[IsEnabled]	[SelectedValue]	[PrimaryTable]	[LimitColumn]	[UseAsQuickFilter]
21		[Variable]	Standard	Filter Report	AdvancedFilter	False	True		GL2018		TRUE
22											

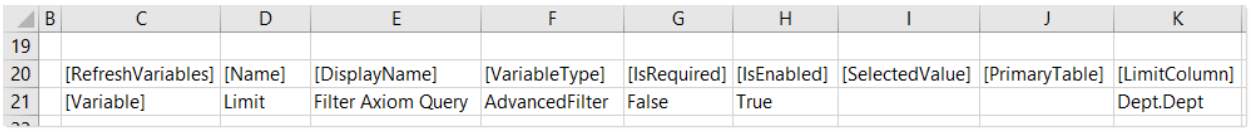
Variable-specific properties (limit query creation)

The following additional properties apply to AdvancedFilter variable types, when using the variable to create a limit query statement. In this case, the only valid use for the selected value is within the **Limit query data based on another table** setting of an Axiom query. For more information about using limit queries, see Limit the data in an Axiom query based on another query.

Column Tag	Description
[LimitColumn]	The column to limit in the Axiom query. This column must be present in the Axiom query. You can use fully qualified Table.Column syntax or just a column name: • If a Table.Column name is specified, then that table is the primary table for the wizard. This means that you do not need to complete the [PrimaryTable] field. When the limit query statement is created, the full Table.Column name will be listed in the Limit parameter. • If only a column name is specified, then you must complete the [PrimaryTable] field. When the limit query statement is created, only the column name will be listed in the Limit parameter.
[PrimaryTable]	Optional. The primary table to determine the table choices available in the Filter Wizard. This selection determines the tables available to limit the data in the Axiom query, based on the specified limit column. The user will first select a table, and then define a filter based on the selected table. If the limit column uses Table.Column syntax, then that table is the primary table, and this property should be left blank. If the limit column is just a column name, then you must enter the name of the primary table here. The user will be able to select any table with a lookup to the limit column (or the primary table itself). Note that if the limit column is something like GL2020.Dept (where this column looks up to Dept.Dept), the primary table is assumed as Dept.
[PlaceholderText]	Optional. Defines placeholder text to display within the variable box until a value is selected. This text also displays as a tooltip for the **Select Filter** button.

The following properties do not apply to AdvancedFilter variables when used to create a limit query: ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, Hierarchies, DataSourceName, DisplayFormat, MinDate, MaxDate, TooltipColumn, AutoQuoteString, UseAsQuickFilter, MinValue, MaxValue, StepFrequency.

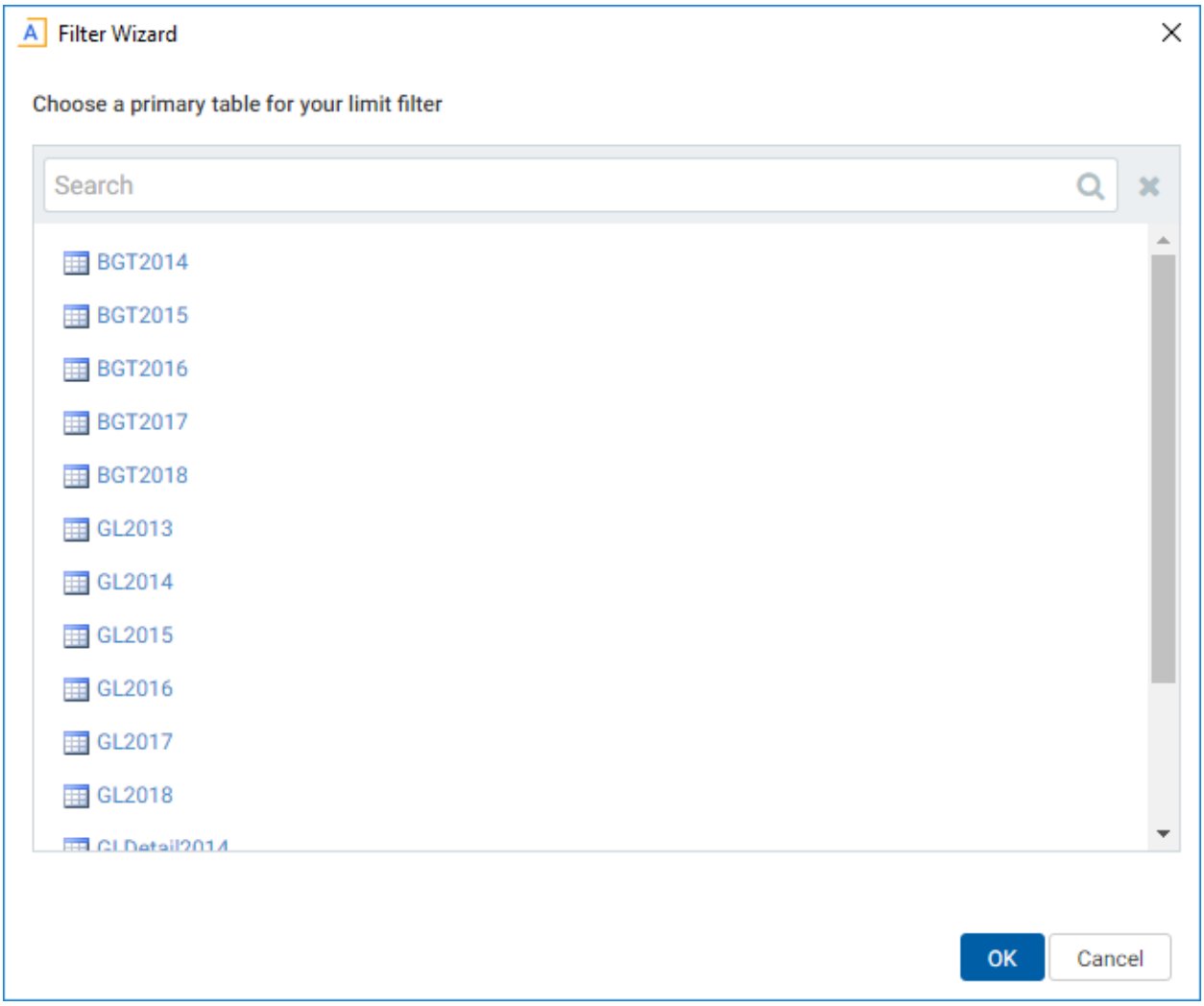
The following screenshot shows an example AdvancedFilter variable to create a limit query statement:



	B	C	D	E	F	G	H	I	J	K
19										
20		[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsRequired]	[IsEnabled]	[SelectedValue]	[PrimaryTable]	[LimitColumn]
21		[Variable]	Limit	Filter Axiom Query	AdvancedFilter	False	True			Dept.Dept

The Filter Wizard dialog and selected value are different when using the AdvancedFilter refresh variable to create a limit query statement. When the user clicks the button to create the filter, they are presented with a two-screen wizard.

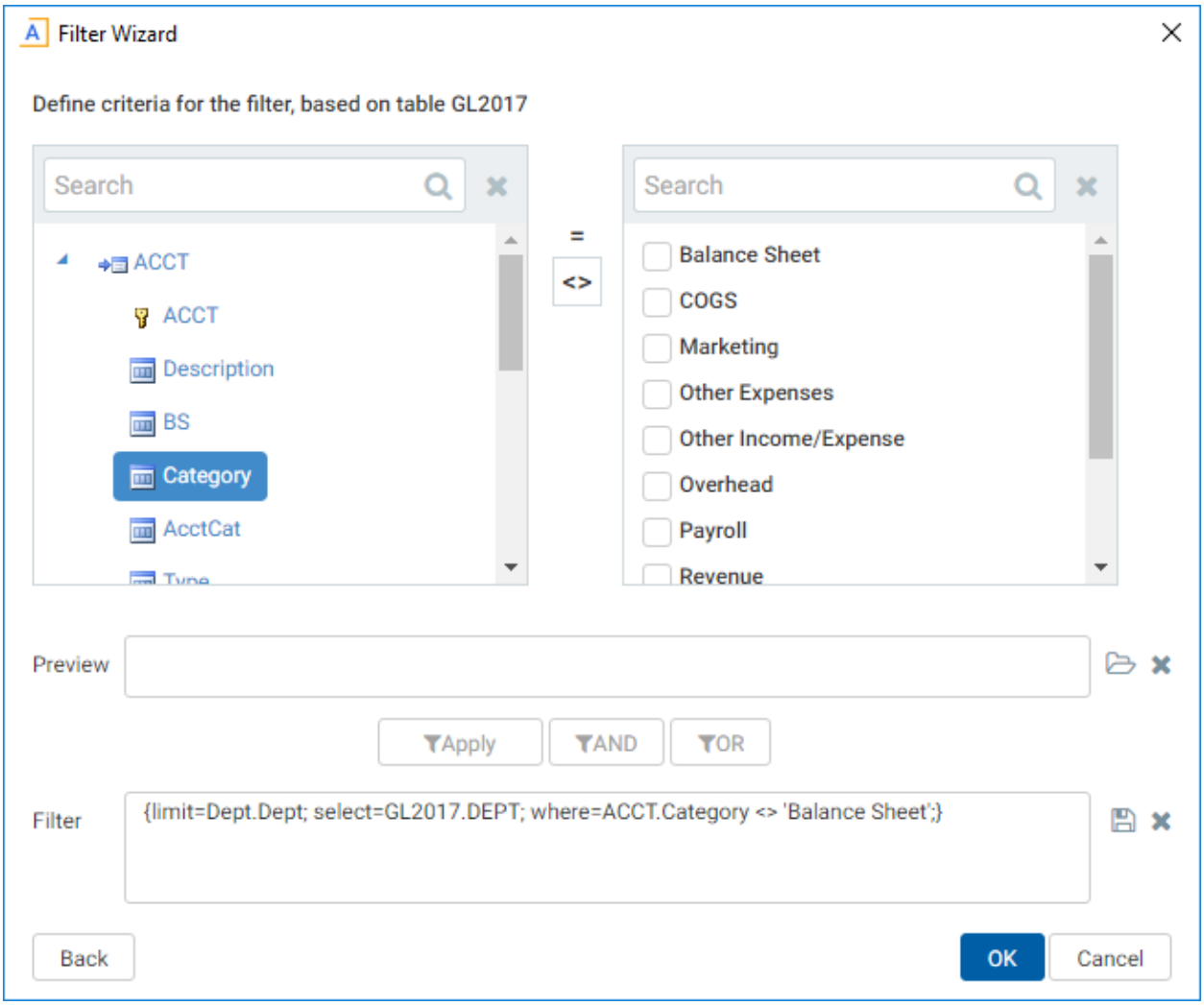
The first screen asks the user to select a table for the limit query. This list is generated based on the table for the designated limit column in the variable properties. The user's selection here becomes the Select column for the limit query statement, and determines the available tables for the filter.



In our example, the limit column for the variable is Acct.Acct. This means that the first screen of the wizard shows all tables with a single-path lookup to Acct.Acct. If the user selects GL2017, the Select column is GL2017.Acct.

NOTE: If a table has multiple paths to the limit column, that table does not display in the wizard. For example, if the limit column is Physician.Physician, and table Encounter has two different columns that use Physician.Physician as a lookup column, then table Encounter is not available in the wizard.

As soon as the user selects a table, the wizard progresses to the second screen to create a filter for the limit query statement. If the user needs to go back and change the table, they can click the Back button at the bottom of the screen.



The second screen contains the advanced view of the filter wizard. The list of tables is filtered based on the table selection from the previous screen. Generally speaking, the filter can be created on the selected table or on a lookup reference table. After the user creates a filter and clicks Apply, the Filter box populates with a limit query statement that includes the filter.

NOTE: Currently, the user must create a filter in order to populate the Filter box with the limit query statement. However, limit query statements are not required to contain a filter. If the user does not want a filter, they can manually remove the Filter parameter from the statement after it has been created.

When the user clicks OK, the limit query statement is written back to the `[SelectedValue]` field, where it can be referenced by the **Limit query data based on another table** option of an Axiom query. This is the only valid place to use a limit query statement.

Calendar refresh variable

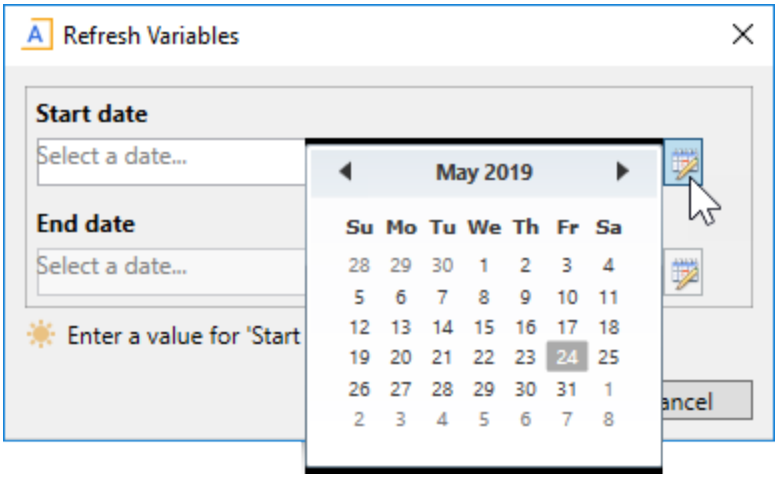
Calendar refresh variables prompt users to specify a date from a calendar. The date is written back to the data source, where it can be used to impact data queries.

For example, you can use the calendar variable to prompt users to select a start date for the data in a report. The date can then be used in a filter that affects the data refresh.

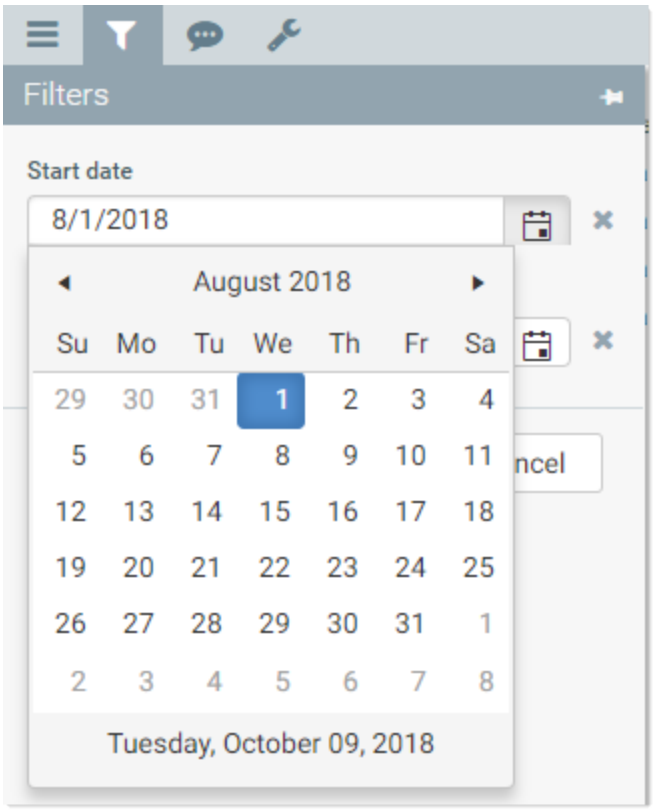
When used in the Web Client, calendar variables can also be used to select a month or year.

Variable behavior

The variable displays as a text box with a calendar button next to it. The user can click the button to select a value from the calendar.



Desktop Client: Example Calendar refresh variable

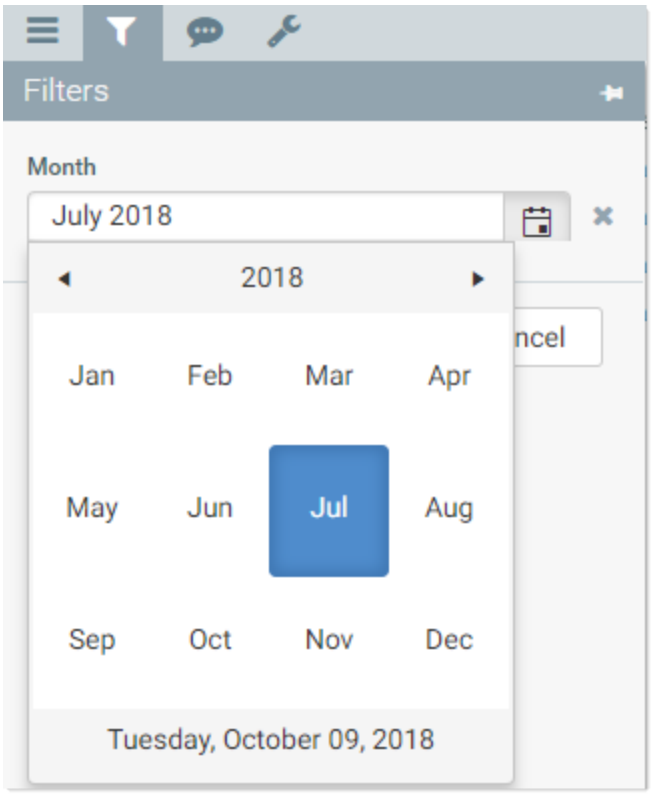


Web Client: Example Calendar refresh variable

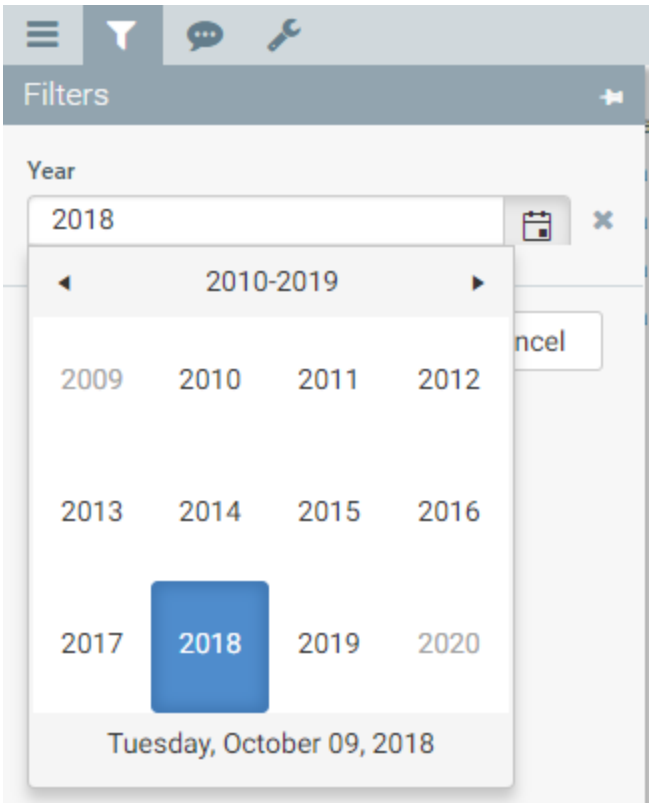
In the Web Client, the current date shows at the bottom of the calendar for reference.

Once the user has selected a date from the calendar, the selected date displays in the text box. The user cannot type a date into the text box; only selection from the calendar is allowed.

In the Web Client only, Calendar refresh variables can also be used to select a month/year combination, or a year. The month and year options are not available in the Desktop Client.



Web Client: Example Calendar refresh variable to select month/year



Web Client: Example Calendar variable to select year

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a Calendar variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define Calendar variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

The following additional properties apply to Calendar variable types:

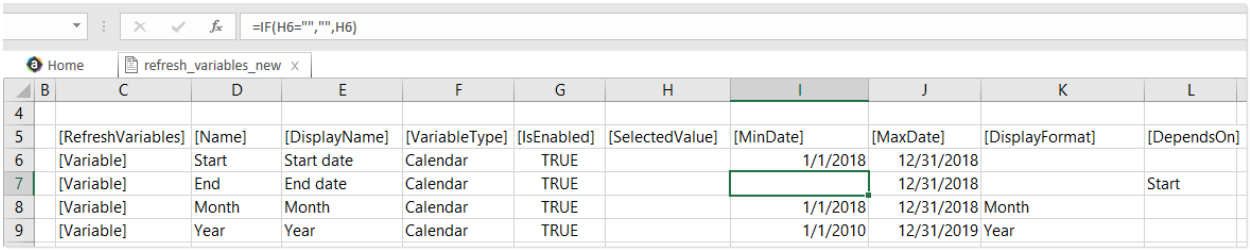
Column Tag	Description
[MinDate]	Optional. The earliest date that is valid for a user to select in the calendar. If specified, the calendar control will not allow the user to select a date that is earlier than this date. When using the Month or Year display format, the minimum date must still be a full date. The appropriate minimum month and year will be determined from that date. **NOTE:** The cell format for this property should be set to Date. DateTime formats are not supported and may cause errors. For more information, see Handling date formats.
[MaxDate]	Optional. The latest date that is valid for a user to select in the calendar. If specified, the calendar control will not allow the user to select a date that is later than this date. When using the Month or Year display format, the minimum date must still be a full date. The appropriate minimum month and year will be determined from that date. **NOTE:** The cell format for this property should be set to Date. DateTime formats are not supported and may cause errors. For more information, see Handling date formats.
[DisplayFormat]	Optional, applies to Web Client only. Specifies the type of date value for selection: • **Date**: Users select specific dates from a calendar control. This is also the default behavior if no display format is specified. • **Month**: Users select a month and year combination from a drop-down selection. • **Year**: Users select a year from a drop-down selection. The display format determines the values for selection and the display of the selected value in the variable. However, the return value written to the [SelectedValue] field is always a full date. See Handling date formats for more information.
[AutoQuoteString]	Optional. Specifies whether the date value is placed in single quotation marks when it is written to the [SelectedValue] column (True/False). If omitted or blank, the default setting is False, which means the date value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value must be wrapped in single quotation marks. For example: Request.Date='2/1/2020'.

The following properties do not apply to Calendar variables: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, Hierarchies, UseAsQuickFilter, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows example Calendar variables.

- The first two variables define start and end dates. The End variable is dependent on the Start variable, so that the minimum value of the end date can use a formula that points to the selected value of the start date. This is to ensure that the user does not accidentally set the end date to an earlier value than the start date.
- The third and fourth variables use the `[DisplayFormat]` to select a month/year combination and a year. This configuration is only valid for use in the Web Client; it will be ignored in the Desktop Client.



	B	C	D	E	F	G	H	I	J	K	L
4											
5		[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsEnabled]	[SelectedValue]	[MinDate]	[MaxDate]	[DisplayFormat]	[DependsOn]
6		[Variable]	Start	Start date	Calendar	TRUE		1/1/2018	12/31/2018		
7		[Variable]	End	End date	Calendar	TRUE			12/31/2018		Start
8		[Variable]	Month	Month	Calendar	TRUE		1/1/2018	12/31/2018	Month	
9		[Variable]	Year	Year	Calendar	TRUE		1/1/2010	12/31/2019	Year	

Handling date formats

Within the Calendar refresh variable, the selected values are displayed as follows, depending on the specified `[DisplayFormat]`. The exact format depends on your system locale.

- **Date**: Dates are displayed in an Excel "short date" format, such as 1/1/2020.
- **Month**: Months are displayed in month/year format, such as January 2020. Only applies to Web Client.
- **Year**: Years are displayed as the year number, such as 2020. Only applies to Web Client.

However, in all cases, the selected value is written to the `[SelectedValue]` field as an Excel date serial number. If you want to use just the selected month or the year in your calculations, you may need to use functions such as MONTH or YEAR to extract the information from the full date.

If you want to set a default value for the refresh variable (or set minimum and maximum dates to restrict the valid selections), then your value must be resolvable as an Excel date value. For example, if you want to set a default value when using the Year selection type, you cannot simply enter the number 2020. Instead, you must enter a date such as 1/1/2020. The refresh variable will resolve this date value as the year 2020. If you enter just the number, then the refresh variable will not interpret that value as a date.

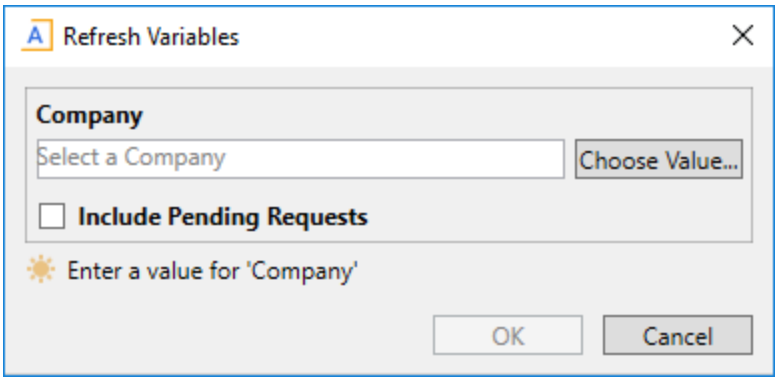
CheckBox refresh variable

CheckBox refresh variables prompt users to enable or disable something by using a check box. The value True (checked) or False (unchecked) is written to the RefreshVariables data source. This value can then be used to enable or disable something that impacts the data refresh.

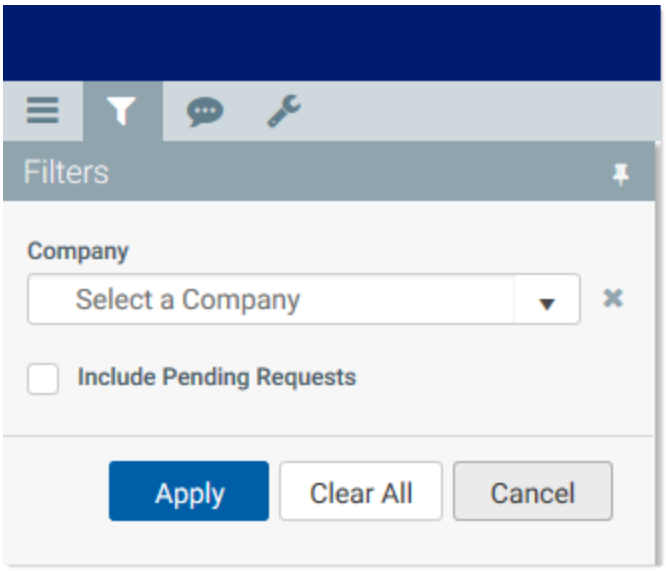
For example, you can use the check box to prompt users to indicate whether a certain set of data should be included in a report. Imagine that you have a report that always includes approved capital requests, but you want to give users the option to also include pending capital requests. The user can use the check box to include the pending requests or not. If the selection is binary, a check box may be more effective than a drop-down list.

Variable behavior

The variable displays as a check box. The user can check or clear the check box.



Desktop Client: Example CheckBox refresh variable



Web Client: Example CheckBox refresh variable

If the variable is optional and it starts out with no value—meaning blank for `[SelectedValue]`—then the value False will *not* be written to the data source if the user leaves the check box unselected. Instead, the selected value is left as blank. To avoid this blank case, it is recommended to configure CheckBox variables as required. This forces the current value of the check box to be written to the data source, even if the user does not interact with the check box. Additionally (or alternatively), you can define a default value for the check box in the `[SelectedValue]` field, so that it starts out as True or False instead of blank.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a CheckBox variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define CheckBox variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

CheckBox variables do not have any variable-specific properties. Only the general variable properties are available.

The following properties do not apply to CheckBox variables: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, AutoQuoteString, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows an example CheckBox variable. In this example, the CheckBox variable is required, and a default value of False has been defined for the variable, so that it starts off as unchecked. This configuration is intended to ensure that the variable never has a blank value, so the blank case does not need to be handled in formulas.

	A	B	C	D	E	F	G	H	I
10									
11		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[ColumnName]	[PlaceholderText]
12		[Variable]	Company	Grid	TRUE		TRUE	Company.Company	Select a Company
13		[Variable]	Include Pending Requests	Checkbox	TRUE	FALSE	TRUE		

ComboBox refresh variable

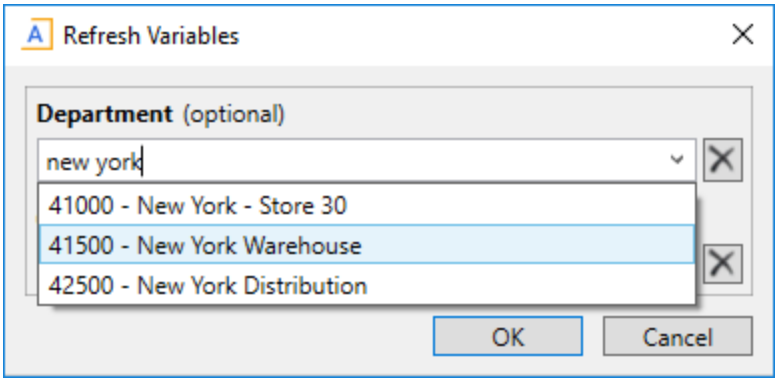
ComboBox refresh variables prompt users to select a value from a searchable drop-down list (the combo box). There are several different options to define the list of values for the combo box:

- **Using a ComboBox data source defined within the file.** ComboBox data sources support defining separate labels and values. This means that what displays to end users in the combo box can be different than what is placed in the `[SelectedValue]` column for use in impacting the data query. For example, end users might select "friendly" country names (the labels) but the corresponding country code (the values) is placed in the `[SelectedValue]` column.
- **Using a table column.** You can specify a table column to display the values from that column in the drop-down list.
- **Using a picklist table.** You can specify a picklist table to display the values from that picklist in the drop-down list. This option provides special support to automatically display the picklist value to users, but write back the corresponding code as the selected value.
- **Using an Axiom Query.** You can define an Axiom query in the file, and designate this query as the source for the combo box. The values returned by the query display in the drop-down list. The query runs "in the background" when the user interacts with the combo box.

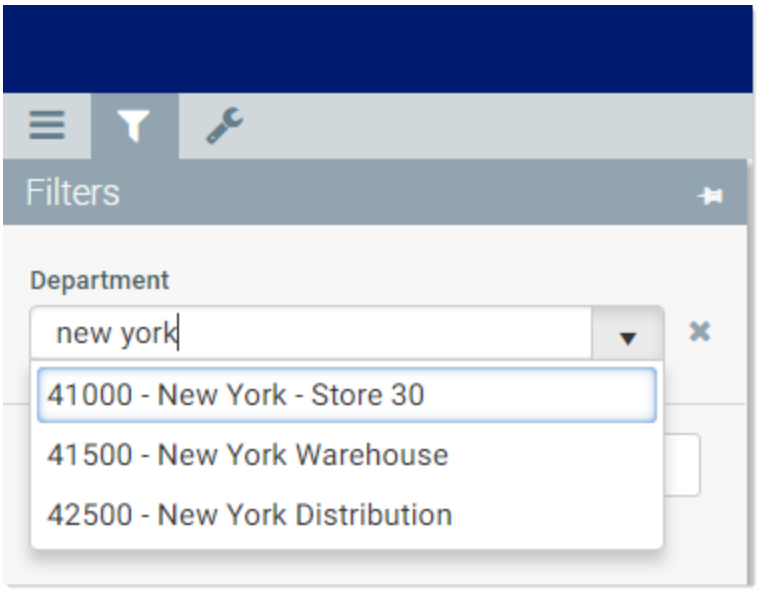
When using a ComboBox with a table column or an Axiom query, you can also use RelatedColumnValue variables. These are special variables that do not display to users. Instead, they are used to return values from related columns so that these values can be referenced in titles and other areas of the file. For example, if you are prompting users to select an account, then you can use a RelatedColumnValue variable to return the description of that account from the Acct.Description column. For more information, see RelatedColumnValue refresh variable.

Variable behavior

The variable displays as a drop-down list with a searchable entry box. The user can scroll the list and select the value directly, or type into the box to find a particular value.



Desktop Client: Example ComboBox refresh variable



Web Client: Example ComboBox refresh variable

The drop-down lists may be limited to displaying some number of values (the specific display limit depends on the source of data and the client environment). For example, in some cases the drop-down list will only display the first 100 values. However, all values can be found by using the search box. The search matches on the primary value, any description columns, and any additional columns included in the display format.

The values in the drop-down list are sorted as follows:

- **ComboBox data source:** Values are presented in the same order as the data source.
- **Table column:** Values are sorted based on the display format if defined, otherwise based on the value column.
- **Picklist table**: Values are sorted based on the string value of the picklist by default. If a display format is defined, values are sorted based on the display format.
- **Axiom query:** Values are sorted according to the Axiom query **Data Sort** setting.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a ComboBox variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define ComboBox variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties (ComboBox data source)

The following additional properties apply to ComboBox variable types, when using a ComboBox data source:

Column Tag	Description
`[DataSourceName]`	The name of the ComboBox data source to provide the list of values for the variable. You must define a ComboBox data source within the file in order to use this option. The name of the data source is defined within the ComboBox tag. For example, if the tag is `[ComboBox;MyName]`, then you would enter `MyName` as the data source name. The ComboBox data source has two property columns: `[Label]` and `[Value]`. The labels define the list of values that display to users. When a user selects a label, the corresponding value is placed in the `[SelectedValue]` column. For more information, see Creating a ComboBox data source for the variable.
`[PlaceholderText]`	Optional. Specifies placeholder text to display within the combo box until a value is selected. If blank, then the default text "Choose a value" is used.
`[AutoQuoteString]`	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. **NOTE:** The values in the ComboBox data source are always considered to be strings and will be quoted if this option is enabled, even if the values are actually numbers.

The following properties do not apply to ComboBox variables when using a ComboBox data source: ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (table column)

The following additional properties apply to ComboBox variable types, when using a table column:

Column Tag	Description
`[ColumnName]`	The column to provide the list of values for the variable. Enter a fully-qualified Table.Column name such as `Acct.Acct`. Multi-level lookups can be used. You can specify any column from any client table in your system. System tables such as Axiom.Aliases are not supported for use with 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 `GL2020.Dept`, this is the same as specifying `GL2020.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 (GL2020 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, `GL2020.Dept` returns only the departments used in the GL2020 table, whereas `Dept.Dept` returns the full list of departments defined in the Dept table. If the value column is a key column or a validated column, then the corresponding descriptions automatically display with the column values in the drop-down list, unless a display format is defined.
`[ColumnFilter]`	Optional. Specifies a filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**. 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.
`[PlaceholderText]`	Optional. Specifies placeholder text to display within the combo box until a value is selected. If blank, then the default text "Choose a value for *ColumnName*" is used.

Column Tag	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 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. If a display format is defined, the items in the list are sorted based on the display format instead of the value column. **NOTE:** This is the only way to display additional columns in the list. The [AdditionalColumns] property does not apply to ComboBox variables. Additional columns included in the display format are searchable within the list.
[TooltipColumn]	Optional. Specifies a column that defines tooltip text for each value shown in the list. When a user hovers over a value in the list, the corresponding text from this column is shown in a tooltip. If the value column uses a lookup, then the tooltip column must be resolvable from the primary table, or must use a fully qualified path from the starting table.
[AutoQuoteString]	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the [SelectedValue] column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: Dept.VP='Smith'. **NOTE:** This option only applies if the value column is a string column.

The following properties do not apply to ComboBox variables when using a table column: ListChoices, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (Picklist table)

Although you can specify a picklist table column using the `[ColumnName]` field, the `[DataSourceName]` field provides special support for picklist tables. You can specify just the picklist table name, and the combo box will automatically display the picklist value to users for selection, while using the corresponding code as the selected value.

Column Tag	Description
`[DataSourceName]`	Specifies the picklist table to define the list of values. Use the following syntax: `Picklist: TableName` For example, Picklist: Category uses the picklist table named Category. When a picklist table is the data source, this is effectively the same as designating the Code column of the picklist table as the source column. However, instead of displaying the integer codes to users, by default the drop-down list displays the string values to users. Users select a string value from the list, but the selected value is the corresponding code.
`[ColumnFilter]`	Optional. Specifies a filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**.
`[PlaceholderText]`	Optional. Specifies placeholder text to display within the combo box until a value is selected. If blank, then the default text "Choose a value for *ColumnName*" is used.

Column Tag	Description
[DisplayFormat]	Optional. Defines a display format for the items in the list. By default, items in the list are displayed using just the value in the Value column of the designated picklist table. The code and description are not displayed. 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: {Category.Value} - {Category.Description} This would display items in the following format: New - Use this category for new requests The display format can use any column from the picklist table. If a display format is defined, the items in the list are sorted based on the display format instead of the value column.
[TooltipColumn]	Optional. Specifies a column in the picklist table that defines tooltip text for each value shown in the list. When a user hovers over a value in the list, the corresponding text from this column is shown in a tooltip. For example, you could specify Description to display the contents of the Description column as tooltips.

The following properties do not apply to ComboBox variables when using a picklist table: ListChoices, AdditionalColumns, AllowMultiSelect, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, AutoQuoteString, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (Axiom query)

The following additional properties apply to ComboBox variable types, when using an Axiom query:

Column Tag	Description
[DataSourceName]	The name of the Axiom query to provide the list of values for the variable. The sheet where the query is defined must also be specified, for example: Sheet2!AQList For more information on how to set up this query for use with a refresh variable, see Setting up an Axiom query for the variable.

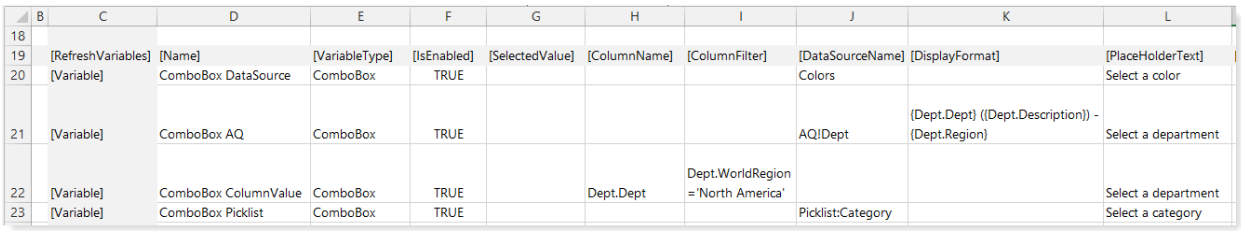
Column Tag	Description
[ColumnFilter]	Optional. A filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**. This property can be used in addition to (or instead of) the **Data Filter** on the Axiom query itself.
[PlaceholderText]	Optional. Specifies placeholder text to display within the combo box until a value is selected. If blank, then the default text "Choose a value for *ColumnName*" is used (where *ColumnName* is the first column in the Axiom query's field definition).
[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 additional columns included here must also be in the field definition of the Axiom query. **NOTE:** This is the only way to display additional columns in the combo box. The [AdditionalColumns] property does not apply to ComboBox variables. Additional columns included in the display format are searchable within the list.
[TooltipColumn]	Optional. Specifies a column that defines tooltip text for each value shown in the list. When a user hovers over a value in the list, the corresponding text from this column is shown in a tooltip. The specified column must also be present in the Axiom query field definition.

Column Tag	Description
[AutoQuoteString]	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the [SelectedValue] column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: Dept.VP='Smith'. **NOTE:** This option only applies if the value column of the Axiom query (the first column in the field definition) is a string column.

The following properties do not apply to ComboBox variables when using an Axiom query: ListChoices, ColumnName, AdditionalColumns, AllowMultiSelect, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows examples of ComboBox variables, one of each type.



	B	C	D	E	F	G	H	I	J	K	L
18											
19		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[ColumnName]	[ColumnFilter]	[DataSourceName]	[DisplayFormat]	[PlaceHolderText]
20		[Variable]	ComboBox DataSource	ComboBox	TRUE				Colors		Select a color
21		[Variable]	ComboBox AQ	ComboBox	TRUE				AQ!Dept	{Dept.Dept} ({Dept.Description}) - {Dept.Region}	Select a department
22		[Variable]	ComboBox ColumnValue	ComboBox	TRUE		Dept.Dept	Dept.WorldRegion ='North America'			Select a department
23		[Variable]	ComboBox Picklist	ComboBox	TRUE				Picklist:Category		Select a category

Creating a ComboBox data source for the variable

If you are defining refresh variables for use in a form-enabled file, then you can use the data source wizard to add the data source. Right-click in the cell where you want to start the data source, then select **Create Axiom Form Data Source > Combo Box**.

If you are defining refresh variables for use in a spreadsheet Axiom file, then you must manually create the data source.

The tags for the Combo Box data source are as follows:

Primary tag

[ComboBox;*DataSourceName*]

The DataSourceName identifies this 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

[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 `[SelectedValue]` column.

Setting up an Axiom query for the variable

When the 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 is the value column for the variable—meaning the values to be placed in the `[SelectedValue]` column of the RefreshVariables data source. If the value column is a key column, then the second column in the field definition is the description column for the key values.
- The field definition of the query must also contain any additional columns used in the `[DisplayFormat]` property, as well as any columns used by associated RelatedColumnValue variables. 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 variable or use a dependent RelatedColumnValue variable, both of those features require the columns to be fully qualified, and they must match the field definition entries exactly.
- The Axiom query data filter can be used to filter the list of values. If desired, you can also (or alternatively) use the `[ColumnFilter]` property of the RefreshVariables data source.
- Although the query itself must be active, 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 combo box.

- No Axiom query settings that impact the display in the sheet will apply to the combo box. 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. One way to think of it is that the values for the drop-down list are basically the same values that you see when using the **Preview Axiom Query Data** button on the Sheet Assistant.
- System tables such as Axiom.Columns are not supported for use in refresh variables, and cannot be used in the Axiom query.

Grid refresh variable

Grid refresh variables prompt users to select a value from a designated table column. For example, you may want the user to select an account from the Acct.Acct column. The list of values is displayed in a dialog, using a searchable grid.

There are two ways that you can specify the column to use for the Grid variable:

- Specify the table column directly, from any table.
- Specify a Picklist table name. This automatically uses the Code column as the value column, but also automatically displays the Value and Description columns.

Both ComboBox variables and Grid variables can be used to select a value from a table column. The primary difference between the two variable types is the user interface for selecting the value—using either a drop-down list or a grid dialog. Additionally, only Grid variables allow selection of multiple values.

When using Grid variables, you can also use RelatedColumnValue variables. These are special variables that do not display to users. Instead, they are used to return values from related columns so that these values can be referenced in titles and other areas of the file. For example, if you are prompting users to select an account, then you can use a RelatedColumnValue variable to return the description of that account from the Acct.Description column. For more information, see RelatedColumnValue refresh variable.

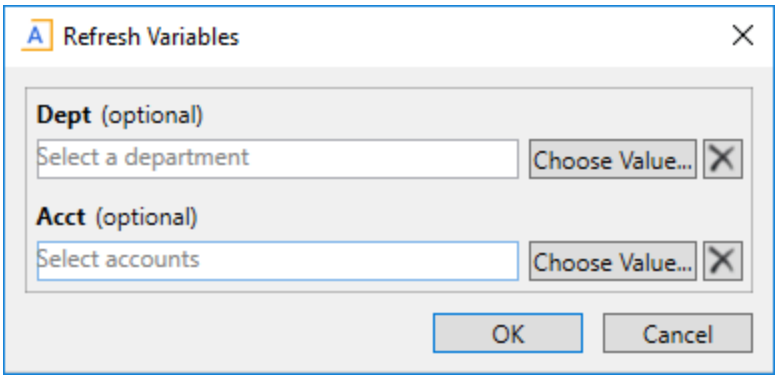
NOTES:

- Grid variables can be used with Axiom forms, but only when multi-select is enabled. If multi-select is not enabled, then the Grid variable will behave like a ComboBox variable, so it is recommended to use a ComboBox variable instead.
- The Grid variable type was formerly known as ColumnValue. You may have older files that still use this legacy name. The ColumnValue variables will continue to work as is, however, you may want to update them to use the new terminology to avoid confusion. If you work on a ColumnValue variable using the Data Source Assistant, it will be treated as a Grid variable.

Variable behavior

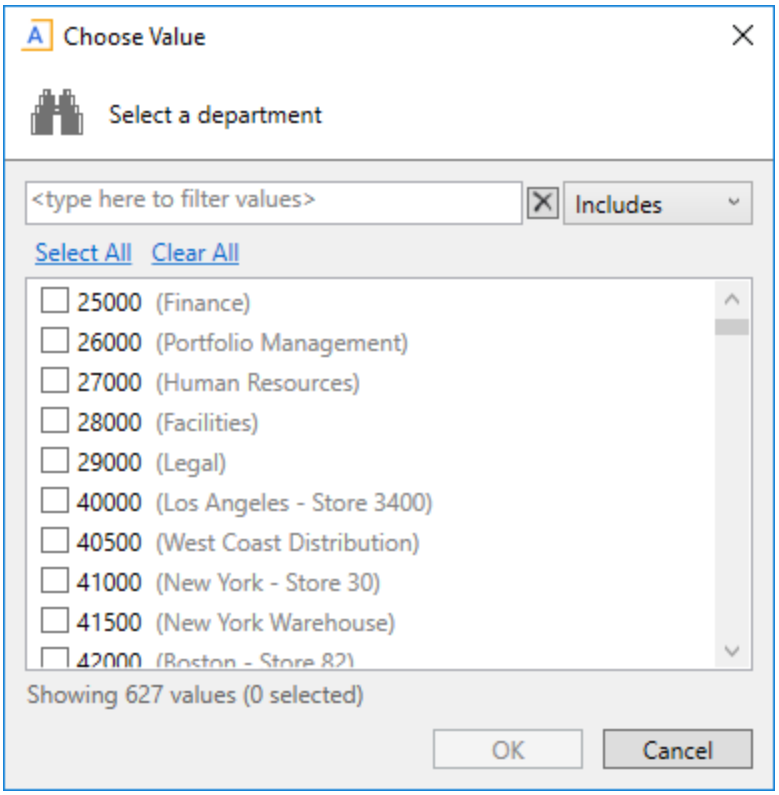
In the Desktop Client, the variable displays as a text box with a **Choose Value** button next to it. The user can do one of the following to specify a value:

- Type a value directly into the text box. The entry must match a value in the designated table column for the variable. If the user enters an invalid value, then a validation message displays at the bottom of the dialog. The **OK** button is disabled until the user enters a valid value.
- Click the button to open the **Choose Value** dialog, to select a value from the designated table column for the variable. This dialog has a search box so that the user can type to find values in lengthy lists. If multi-select is enabled for the variable, then the Choose Value dialog has check boxes to enable multiple selections. Once the user has selected a value, the value displays in the text box.

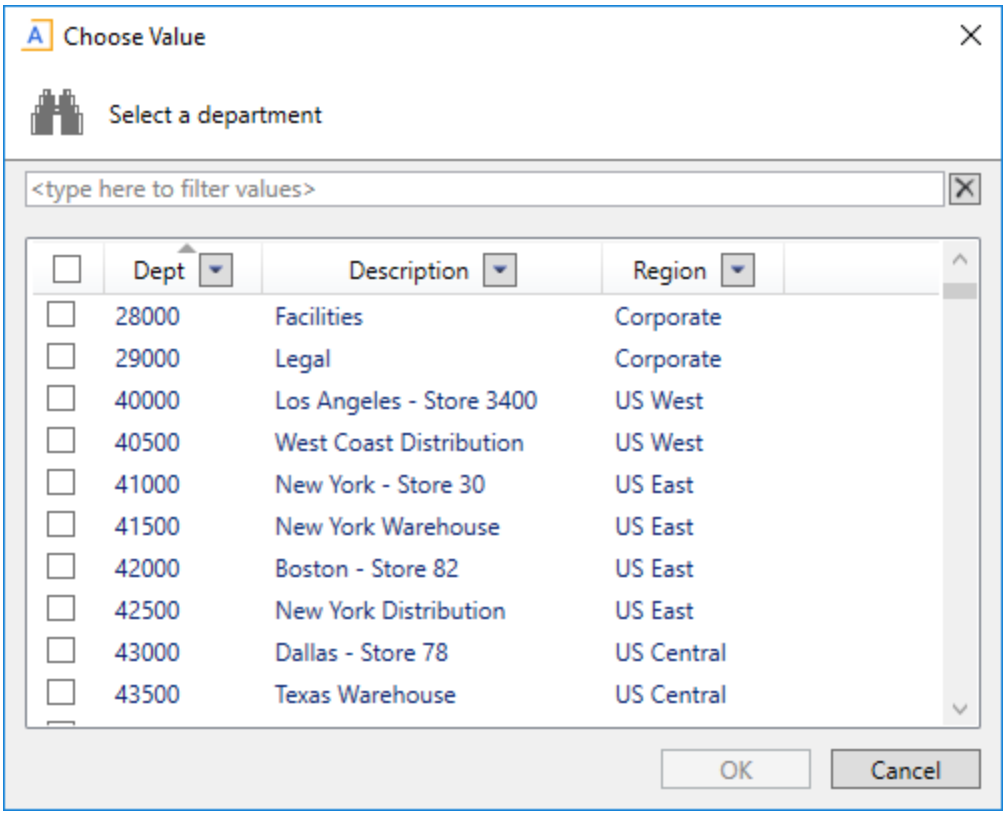


Desktop Client: Example grid variables

The Choose Value dialog uses either the "simple view" or the "full grid view" depending on the data to be displayed in the dialog. If only one column is being shown, or a key column plus description only, then the simple view is used. However, if the target column is a key column and any additional non-description columns are included, then the full grid view is used.



Example Choose Value dialog using "simple view" and with multi-select enabled

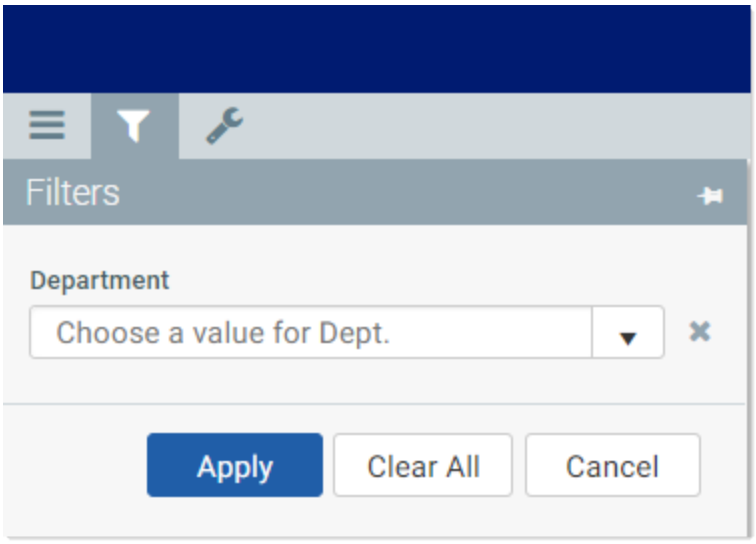


Example Choose Value dialog using "full grid view" and with multi-select enabled

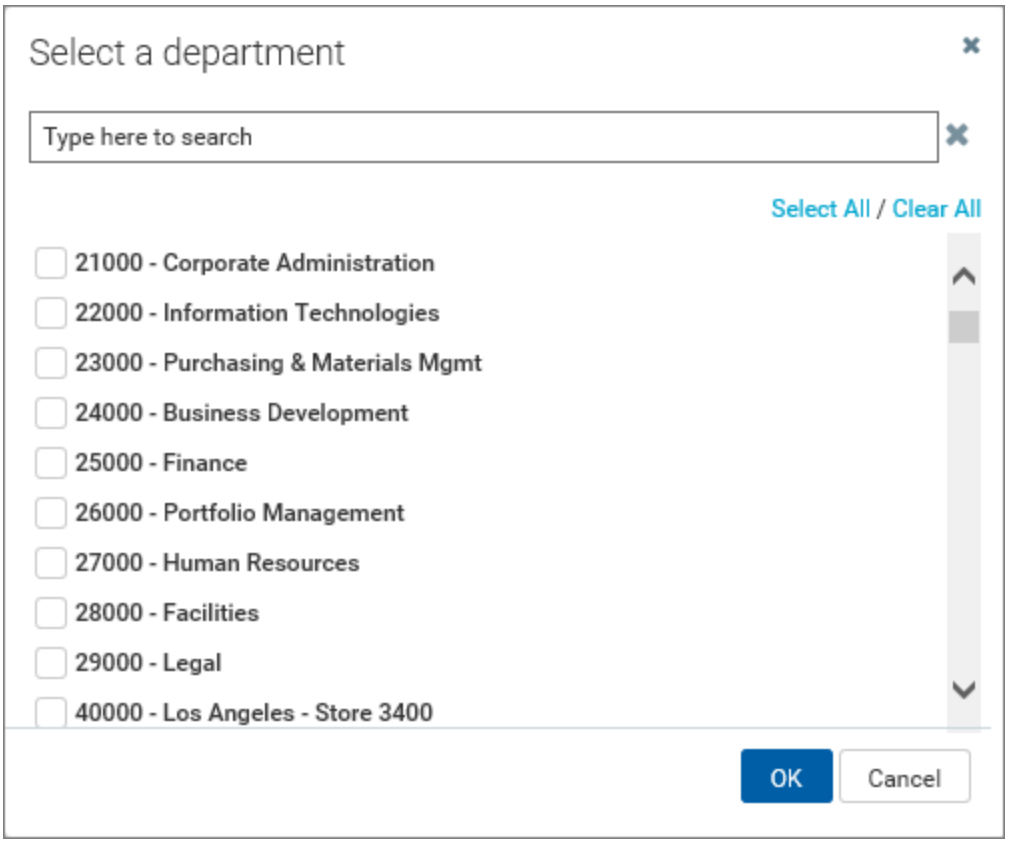
The list of values in the Choose Value dialog is sorted based on the designated table column for the variable. In some cases the values may not be displayed as expected:

- If the column is from a document reference table, the values will most likely not be displayed in the order they are stored in the source file. When the values are saved to the database from the source file, they are sorted based on the key column of the table (column A of the sheet).
- If the column is a string column that contains numeric values (values that are all numbers and values that start with numbers), Axiom will attempt to sort these values in numeric order. Values that start with letters will then be sorted in alphabetical order.

When used with Axiom forms and multi-select is enabled for the variable, the variable initially looks like a ComboBox variable.



However, when the user clicks the down arrow to select a value, a dialog opens instead, allowing the user to select multiple values using check boxes. The behavior of this dialog is the same as when using multi-select with a form combo box.



The "full grid" view is not used with Axiom forms, even if multiple additional columns are specified. The additional column values are concatenated with hyphens. This means that the values cannot be sorted by column, but users can still use the search box to find particular values.

If multi-select is not enabled for the variable, then the variable behaves the same way as a ComboBox variable when used in Axiom forms. Therefore, it is strongly recommended to use a ComboBox variable with Axiom forms instead, if you do not need multi-select.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a Grid variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define Grid variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties (table column)

The following additional properties apply to Grid variable types, when specifying a table column directly.

Column Tag	Description
`[ColumnName]`	The column to provide the list of values for the variable. Enter a fully-qualified Table.Column name such as `Acct.Acct`. Multi-level lookups can be used. You can specify any column from any client table in your system. System tables such as Axiom.Aliases are not supported for use with 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 `GL2020.Dept`, this is the same as specifying `GL2020.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 (GL2020 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, `GL2020.Dept` returns only the departments used in the GL2020 table, whereas `Dept.Dept` returns the full list of departments defined in the Dept table. If the value column is a key column or a validated column, then the corresponding descriptions automatically display with the column values in the grid, unless additional columns are defined.

Column Tag	Description
[AdditionalColumns]	Optional. One or more additional columns to display in the selection dialog along with the value column. Separate multiple column names with commas or semicolons. 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. Column-only syntax is only allowed for columns directly on the primary table. **NOTES:** • If [AdditionalColumns] is left blank, then any description columns associated with the value column are automatically included in the grid. However, if you do specify any additional columns then you must also include the desired description columns. • When used in the Web Client, additional column values are concatenated to the key value using hyphens. They do not display as separate columns. For example, if the key is Dept and additional columns are Description and Country, the value will display like 24000 - Finance - United States. If you do not like this display, then in the Web Client only, you can use the [DisplayFormat] parameter to impact the display of the list in the multi-select dialog.
[ColumnFilter]	Optional. A filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**. 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.
[PlaceholderText]	Optional. Defines placeholder text to display within the variable box until a value is selected. This text also displays at the top of the **Choose Value** dialog, and as a tooltip for the **Choose Value** button. If blank, then the default text "Choose a value for *ColumnName*" is used.

Column Tag	Description
`[AllowMultiSelect]`	Optional. Specifies whether multiple values can be selected from the designated column (True/False). • If `True`, then the list of values will display with check boxes and the user can select more than one item. The values will be placed in the `[SelectedValue]` column as a comma-separated list. If the column is a string column, the values will automatically be wrapped in single quotation marks. • If blank or `False`, then only one value can be selected from the column. This option should be used if you intend to create a filter based on the user's selections, and you want the user to be able to filter on multiple data elements. The filter must use IN syntax, with the comma-separated list placed within the parentheses. For example, if the selected value cell for the variable is Variables!G16, the filter could be built as follows: `="DEPT.DEPT IN ("&Variables!G16&")"` Which would result in the filter: `DEPT.DEPT IN (20000, 21000, 22000)`

Column Tag	Description
[DisplayFormat]	Optional. Defines a display format for the items in the list. This only applies to Grid variables when used in the Web Client, to control the display of items in the multi-select dialog. The display format is ignored in the Desktop Client. 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. If a display format is defined, the items in the list are sorted based on the display format instead of the value column.
[TooltipColumn]	Optional. Specifies a column that defines tooltip text for each value shown in the list. When a user hovers over a value in the list, the corresponding text from this column is shown in a tooltip. If the value column uses a lookup, then the tooltip column must be resolvable from the primary table, or must use a fully qualified path from the starting table.
[AutoQuoteString]	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the [SelectedValue] column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: Dept.VP='Smith'. **NOTE:** This option only applies if the value column is a string column, and only if multi-select is not enabled for the variable. If multi-select is enabled, then string values are always quoted.

The following properties do not apply to Grid variables when using a table column: DataSourceName, ListChoices, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (Picklist table)

The following additional properties apply to Grid variable types, when specifying a Picklist table.

Column Tag	Description
`[DataSourceName]`	Specifies the picklist table to define the list of values. Use the following syntax: `Picklist: TableName` For example, `Picklist: Category` uses the picklist table named Category. When a picklist table is the data source, this is effectively the same as designating the Code column of the picklist table as the source column. The additional picklist columns of Value and Description are also displayed in the grid.
`[ColumnFilter]`	Optional. A filter criteria statement to limit the list of codes displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**.
`[PlaceholderText]`	Optional. Defines placeholder text to display within the variable box until a code is selected. This text also displays at the top of the **Choose Value** dialog, and as a tooltip for the **Choose Value** button. If blank, then the default text "Choose a value for *ColumnName*" is used.
`[AllowMultiSelect]`	Optional. Specifies whether multiple codes can be selected from the picklist (True/False). • If `True`, then the list of codes will display with check boxes and the user can select more than one item. The codes will be placed in the `[SelectedValue]` column as a comma-separated list. • If blank or `False`, then only one code can be selected from the column.

Column Tag	Description
[DisplayFormat]	Optional. Defines a display format for the items in the list. This only applies to Grid variables when used in the Web Client, to control the display of items in the multi-select dialog. The display format is ignored in the Desktop Client. 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) If a display format is defined, the items in the list are sorted based on the display format instead of the value column. When defining a display format for use in the Web Client, you can also optionally use the TooltipColumn property to display tooltip text for each item. This is not applicable in the Desktop Client, because all picklist columns display in the grid in the Desktop Client.
[TooltipColumn]	Optional. Specifies a column in the picklist table that defines tooltip text for each value shown in the list. When a user hovers over a value in the list, the corresponding text from this column is shown in a tooltip. Although you can specify a tooltip column when using a picklist table data source in the Desktop Client, there is typically no reason to do so, because the grid already shows all columns in the picklist table.

The following properties do not apply to Grid variables when using a picklist table: ListChoices, ColumnName, AdditionalColumns, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, AutoQuoteString, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows examples of Grid variables. The first variable uses a filter and does not allow multi-selection. The second variable specifies additional columns to display in the grid, and also allows multi-selection. The third variable uses a picklist table.

	B	C	D	E	F	G	H	I	J	K	L	M
4												
5		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[ColumnName]	[AdditionalColumns]	[ColumnFilter]	[DataSourceName]	[AllowMultiSelect]	[PlaceHolderText]
6		[Variable]	Dept	Grid	TRUE		Dept.Dept		Dept.WorldRegion ='North America'		FALSE	Select a department
7		[Variable]	Acct	Grid	TRUE		Acct.Acct	Description, Category			TRUE	Select accounts
8		[Variable]	Category	Grid	TRUE					Picklist:Category		Select a category

HierarchyFilter refresh variable

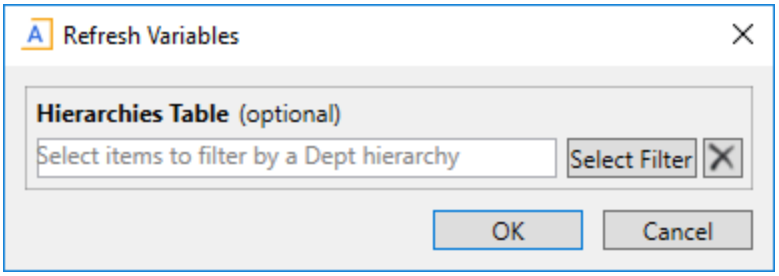
HierarchyFilter refresh variables prompt users to select one or more items from a defined hierarchy. A filter criteria statement is built from the user's selections. This statement can optionally be applied as a Quick Filter during the refresh, or the statement can simply be returned into the `[SelectedValue]` column for the file designer to use as needed.

For example, you can prompt users to select a value from the Geography hierarchy defined on the DEPT table. If the user selects WorldRegion Europe in the hierarchy, this results in a filter criteria statement such as: `DEPT.WorldRegion='Europe'`. The file designer can set up the file so that this statement is referenced in a filter, such as an Axiom query data filter, or the file designer can configure the variable so that the statement is automatically applied as a Quick Filter.

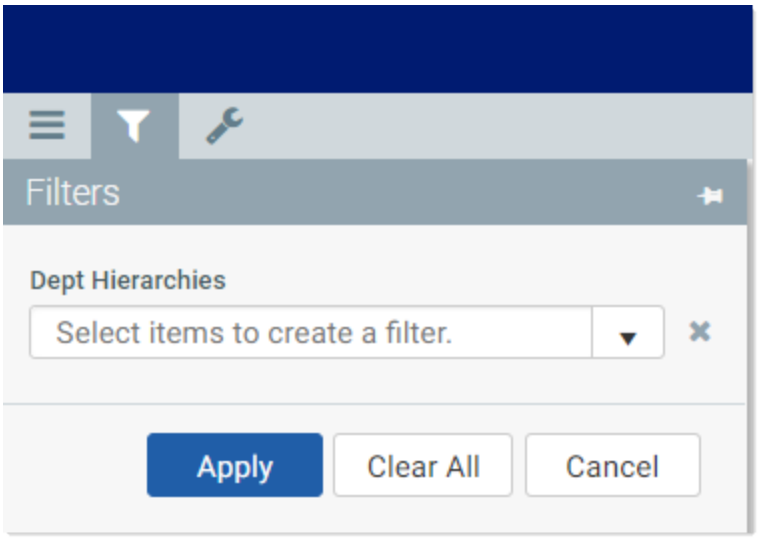
The variable has various options to control which hierarchies are shown to the user, and how the selected hierarchy value is applied.

Variable behavior

The variable displays as a read-only text box with a filter button next to it. The user can click the button to select items in the hierarchy.



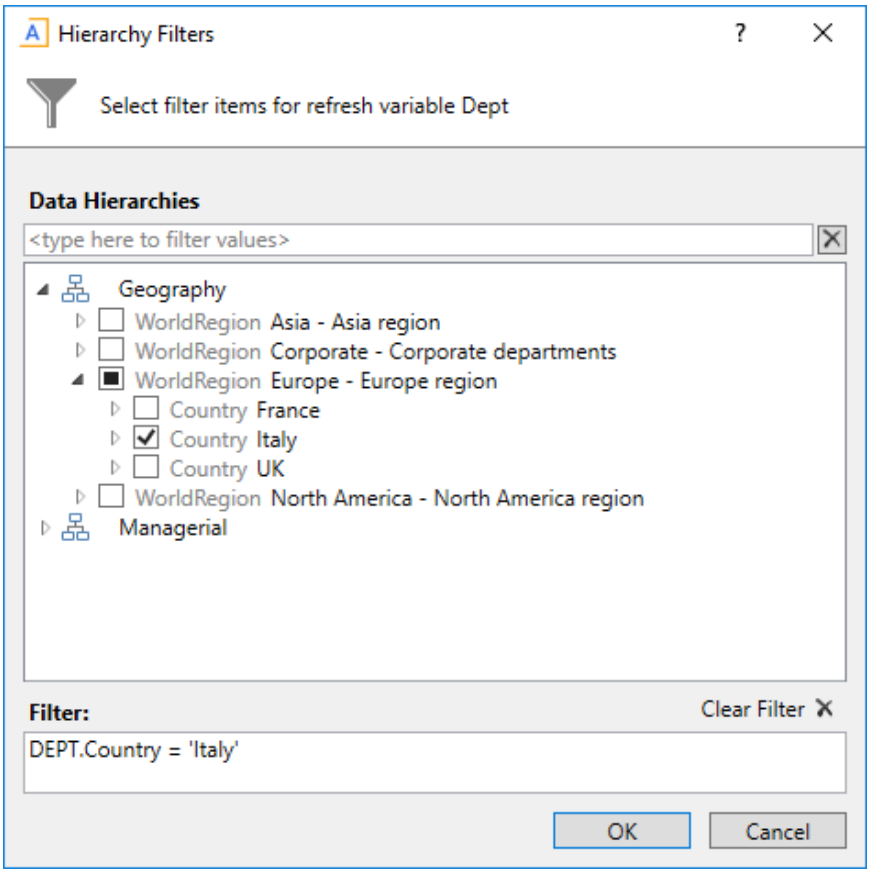
Desktop Client: Example HierarchyFilter refresh variable



Web Client: Example Hierarchy Filter refresh variable

Clicking the **Select Filter** button or the hierarchy icon opens the **Hierarchy Filters** dialog. In this dialog, the user can select the desired hierarchy items to create a filter.

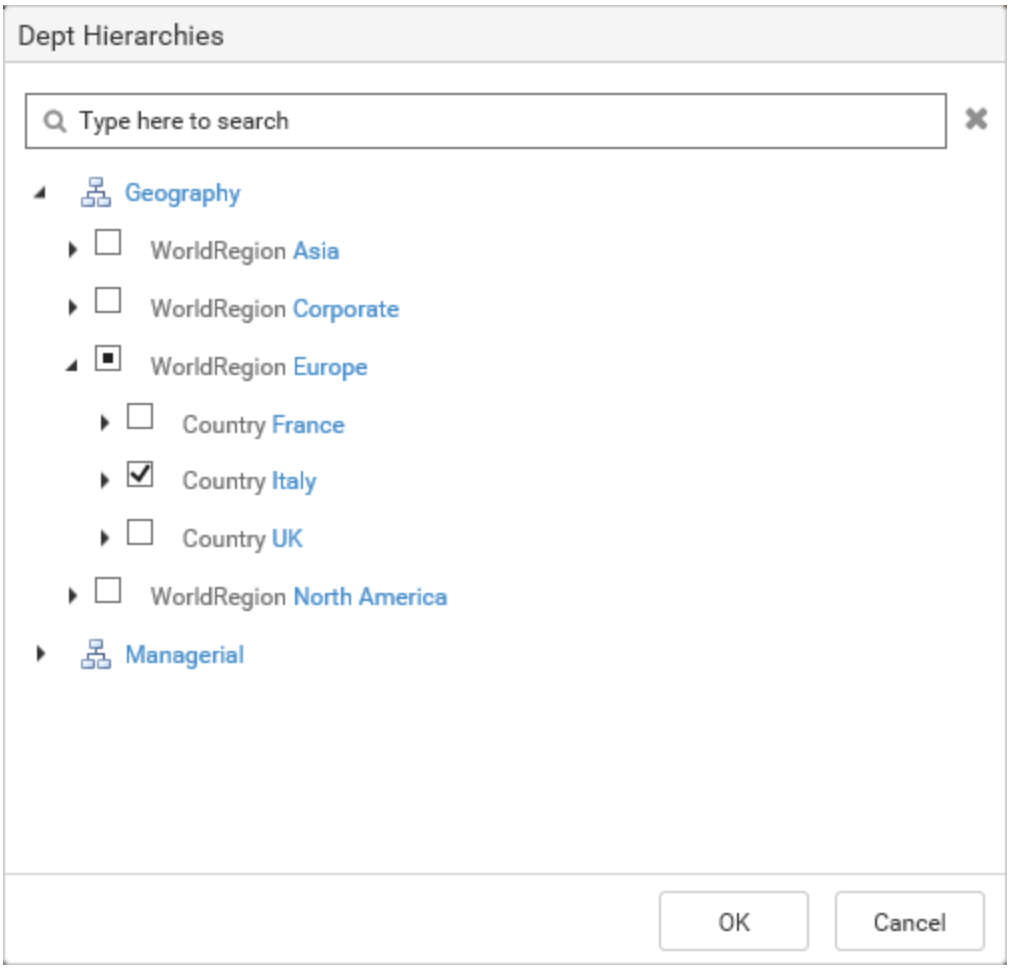
The Hierarchy Filters dialog is similar to the "Simple Filter" view of the Quick Filter dialog in the Desktop Client. The dialog only displays the hierarchies allowed by the refresh variable configuration. Users can select one or more hierarchy items to create a filter. Users can manually expand hierarchy groupings to find items, or they can type into the search box at the top of the dialog to filter the items shown. The "Advanced Filter" view is not available when using a HierarchyFilter refresh variable.



Desktop Client: Example Hierarchy Filters dialog

In the Desktop Client, the filter criteria statement resulting from the user's selections is displayed in the **Filter** box at the bottom of the dialog. The Filter box is not manually editable; the filter can only be created by selecting values from hierarchies. When the user clicks **OK**, the resulting filter displays in the read-only text box for the refresh variable.

In the Web Client, the display of the hierarchy filter is more minimal. In this case, the full filter criteria statement does not display to the user.



Web Client: Example Hierarchy Filters dialog

When the user clicks **OK**, the user's selected hierarchy items display in the read-only text box for the refresh variable. (In this example, the variable would display the text "Italy".) Selections of up to 3 items are displayed in a comma-separated list. If more than 3 items are selected, the variable displays the text "Filter currently applied". The full list of selected items displays in a tooltip.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a HierarchyFilter variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define HierarchyFilter variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

The following additional properties apply to HierarchyFilter variable types:

Column Tag	Description
`[Hierarchies]`	Specifies the hierarchies to display to the user. You can use this field in the following ways: • 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.

Column Tag	Description
[UseAsQuickFilter]	Optional. Specifies whether the filter criteria statement resulting from the hierarchy selection is applied as a Quick Filter. • If True, then the filter criteria statement is applied as a Quick Filter to the workbook. This works just like when using the Quick Filter dialog to apply a temporary filter to the workbook. If a Quick Filter is already applied to the workbook, the old filter is cleared and the new filter replaces it. Only one HierarchyFilter variable can be applied as a Quick Filter per file. **NOTE:** The Quick Filter is applied to the entire workbook. The option to apply a Quick Filter to only the active sheet is not available when applying a Quick Filter via refresh variables. • If blank or False, then the filter criteria statement is not applied as a Quick Filter. If you want to filter data based on this statement, then you must configure the file to do so, such as referencing the statement in the data filter for an Axiom query.
[PlaceholderText]	Optional. Defines placeholder text to display within the variable box until a value is selected. This text also displays as a tooltip for the **Select Filter** button. If blank, then the default text "Select items to create a filter" is used.

The following properties do not apply to HierarchyFilter variables: ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, MinDate, MaxDate, TooltipColumn, AutoQuoteString, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

The following screenshot shows an example HierarchyFilter variable:

	B	C	D	E	F	G	H	I	J
12									
13		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[Hierarchies]	[UseAsQuickFilter]	[PlaceHolderText]
14		[Variable]	Dept Hierarchies	HierarchyFilter	TRUE		Dept	TRUE	Select items to filter by a Dept hierarchy

RadioButton refresh variable

RadioButton refresh variables prompt users to select a value from two or more radio buttons. There are three ways to define the list of values to be displayed as radio buttons. These are the same three options supported by ComboBox variables:

- ComboBox data source
- Table column

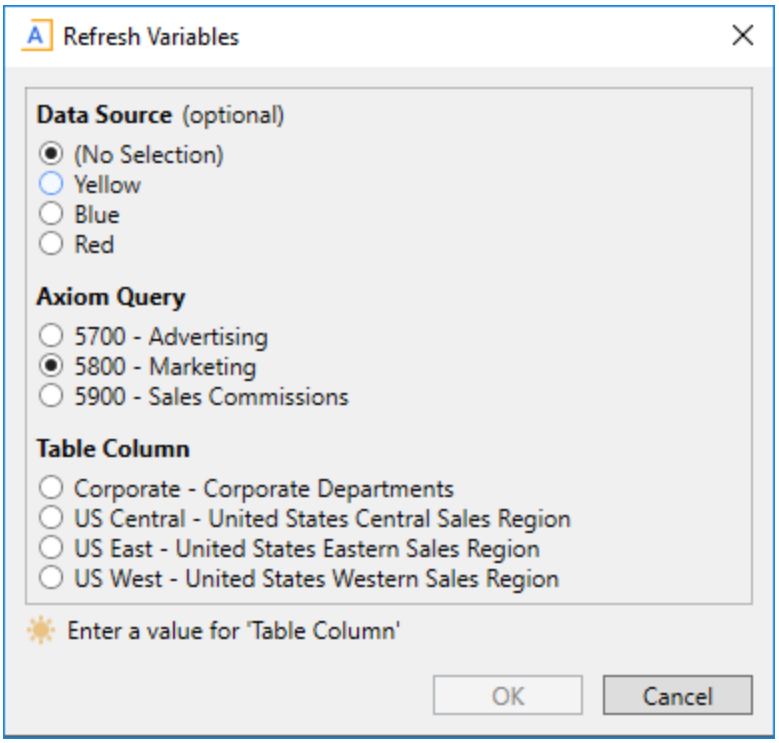
- Axiom query

The RadioButton refresh variable should only be used for small lists of values. Large lists are difficult for users to read in radio button format, and take up too much space in the refresh dialog or filter panel. If the list is too large to display effectively in radio button format, then you should use a different variable type, such as StringList, Grid, or ComboBox.

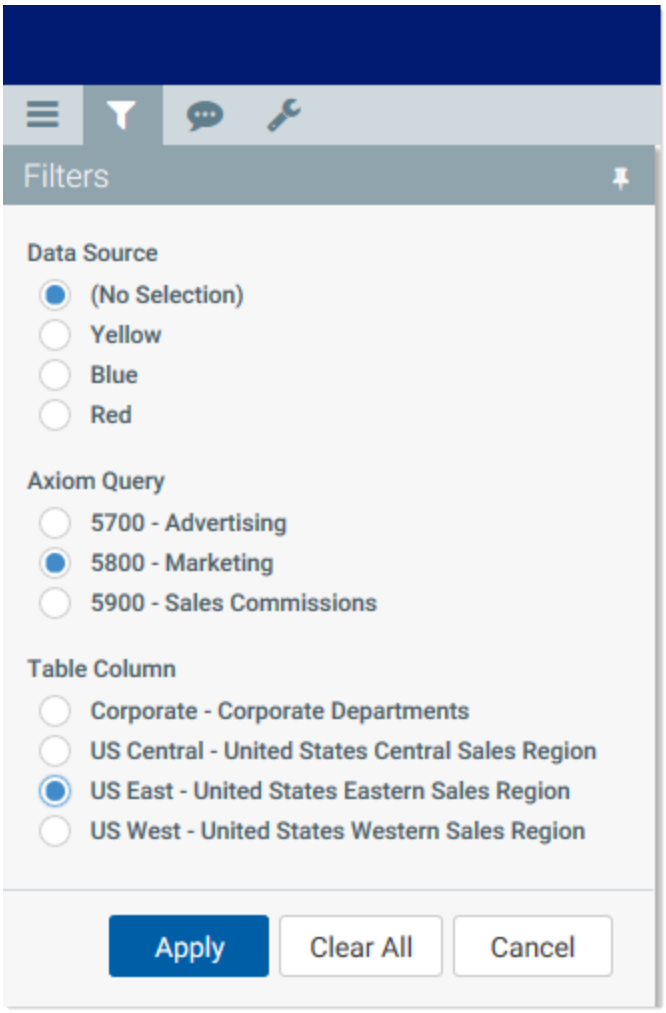
Unlike ComboBox variables, RadioButton refresh variables do not support use of associated RelatedColumnValue variables. This is because the kind of data displayed as radio buttons typically does not have related column values that need to be brought into the spreadsheet. Although ComboBox variables and RadioButton variables use the same data options, the kind of data they use is very different.

Variable behavior

The variable displays as a vertical list of radio buttons. The user can select one of the radio buttons to use that value. The following example shows three different radio button variables, where the first variable is optional and the second two are required.



Desktop Client: Example RadioButton refresh variable



Web Client: Example RadioButton refresh variable

If the variable is not required, the user can select **(No Selection)**. This means that the selected value for the variable will be blank. The (No Selection) option is automatically added to the top of optional variables; you do not need to manually add this value to your list of values. The variable will start out at this value if the variable is not required and no "default value" has been defined.

If the variable is required, then the user must select one of the radio buttons. The (No Selection) option is not available.

The radio button values are presented in the following order, depending on the source data:

- **ComboBox data source:** Values are presented in the same order as the data source.
- **Table column:** Values are sorted based on the display format if defined, otherwise based on the value column.
- **Axiom query:** Values are sorted according to the Axiom query **Data Sort** setting.

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a RadioButton variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define RadioButton variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

The remaining columns depend on whether you are using a ComboBox data source, a table column, or an Axiom query. Because RadioButton variables use small lists of values, the most common way to define the list of values is to use a ComboBox data source. If you use a table column or an Axiom query to define the list of values, you must ensure that the returned list of values is small.

Variable-specific properties (ComboBox data source)

The following additional properties apply to RadioButton variable types, when using a ComboBox data source:

Column Tag	Description
`[DataSourceName]`	The name of the ComboBox data source to provide the list of values for the variable. You must define a ComboBox data source within the file in order to use this option. The name of the data source is defined within the ComboBox tag. For example, if the tag is `[ComboBox;MyName]`, then you would enter `MyName` as the data source name. The ComboBox data source has two property columns: `[Label]` and `[Value]`. The labels define the list of values that display to users. When a user selects a label, the corresponding value is placed in the `[SelectedValue]` column. For more information, see Creating a ComboBox data source for the variable.

Column Tag	Description
`[AutoQuoteString]`	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. **NOTE:** The values in the ComboBox data source are always considered to be strings and will be quoted if this option is enabled.

The following properties do not apply to RadioButton variables when using a ComboBox data source: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (table column)

The following additional properties apply to RadioButton variable types, when using a table column:

Column Tag	Description
[ColumnName]	The column to provide the list of values for the variable. Enter a fully-qualified Table.Column name such as Acct.Acct. Multi-level lookups can be used. You can specify any column from any client 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 GL2020.Dept, this is the same as specifying GL2020.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 (GL2020 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, GL2020.Dept returns only the departments used in the GL2020 table, whereas Dept.Dept returns the full list of departments defined in the Dept table. If the value column is a key column or a validated column, then the corresponding descriptions automatically display with the column values in the drop-down list, unless a display format is defined.
[ColumnFilter]	Optional. A filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**. 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.

Column Tag	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 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. If a display format is defined, the items in the list are sorted based on the display format instead of the value column.
[AutoQuoteString]	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the [SelectedValue] column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: Dept.VP='Smith'. **NOTE:** This option only applies if the value column is a string column.

The following properties do not apply to ComboBox variables when using a table column: PlaceHolderText, ListChoices, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, AutoQuoteString, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Variable-specific properties (Axiom query)

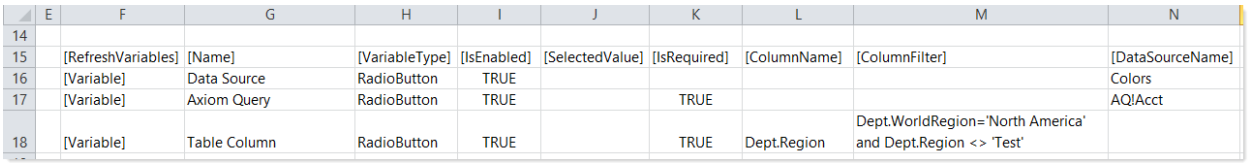
The following additional properties apply to RadioButton variable types, when using an Axiom query:

Column Tag	Description
[DataSourceName]	The name of the Axiom query to provide the list of values for the variable. The sheet where the query is defined must also be specified, for example: `Sheet2!AQList` For more information on how to set up this query for use with a refresh variable, see Setting up an Axiom query for the variable.
[ColumnFilter]	Optional. A filter criteria statement to limit the list of values displayed to the user. You can type in the filter statement manually, or right-click the cell and use **Axiom Wizards > Filter Wizard**. This property can be used in addition to (or instead of) the **Data Filter** on the Axiom query itself.
[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 additional columns included here must also be in the field definition of the Axiom query.
[AutoQuoteString]	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. **NOTE:** This option only applies if the value column of the Axiom query (the first column in the field definition) is a string column.

The following properties do not apply to Radio Button variables when using an Axiom query: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, AllowMultiSelect, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows examples of RadioButton variables, one of each type (data source, Axiom query, and column value).



	E	F	G	H	I	J	K	L	M	N
14										
15		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[ColumnName]	[ColumnFilter]	[DataSourceName]
16		[Variable]	Data Source	RadioButton	TRUE					Colors
17		[Variable]	Axiom Query	RadioButton	TRUE		TRUE			AQ!Acct
18		[Variable]	Table Column	RadioButton	TRUE		TRUE	Dept.Region	Dept.WorldRegion='North America' and Dept.Region <> 'Test'	

Creating a ComboBox data source for the variable

If you are defining refresh variables for use in a form-enabled file, then you can use the data source wizard to add the ComboBox data source. Right-click in the cell where you want to start the data source, then select **Create Axiom Form Data Source > Combo Box**.

If you are defining refresh variables for use in a spreadsheet Axiom file, then you must manually create the data source.

The tags for the Combo Box data source are as follows:

Primary tag

[ComboBox;*DataSourceName*]

The DataSourceName identifies this 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

[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 radio button labeled "Consolidated", then the value "All" will be placed in the `[SelectedValue]` column.

Setting up an Axiom query for the variable

When the RadioButton variable is rendered in the refresh dialog or filter panel, 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 generate the list of radio buttons.

The Axiom query should be set up as follows:

- The first column in the field definition is the value column for the variable—meaning the values to be placed in the `[SelectedValue]` column. If the value column is a key column, then the second column in the field definition is the description column for the key values.
- The field definition of the query must also contain any additional columns used in the `[DisplayFormat]` property. 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 variable, the display format must use fully qualified columns, and they must match the field definition entries exactly.
- The Axiom query data filter can be used to filter the list of values. If desired, you can also (or alternatively) use the `[ColumnFilter]` property of the RefreshVariables data source.
- 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 refresh variable.
- No Axiom query settings that impact the display in the sheet will apply to the variable. 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. One way to think of it is that the values for the drop-down list are basically the same values that you see when using the **Preview Axiom Query Data** button on the Sheet Assistant.

RangeSlider refresh variable

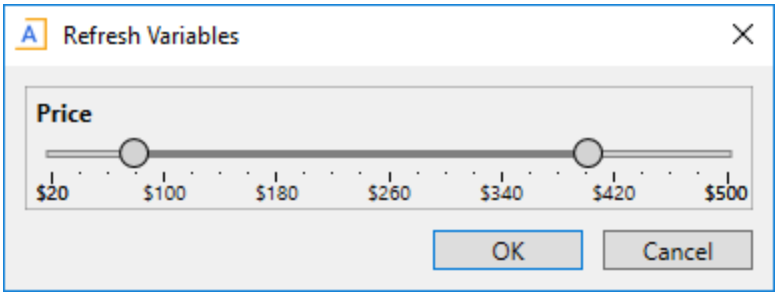
RangeSlider refresh variables prompt users to select top and bottom values within a defined range, using slider handles. The selected values are written back to the data source, where they can be used to impact data queries or other aspects of a report.

For example, you can use the range slider variable to prompt users to select the top and bottom values of a price range. The report can then be filtered to only bring in rows where the price falls within that range.

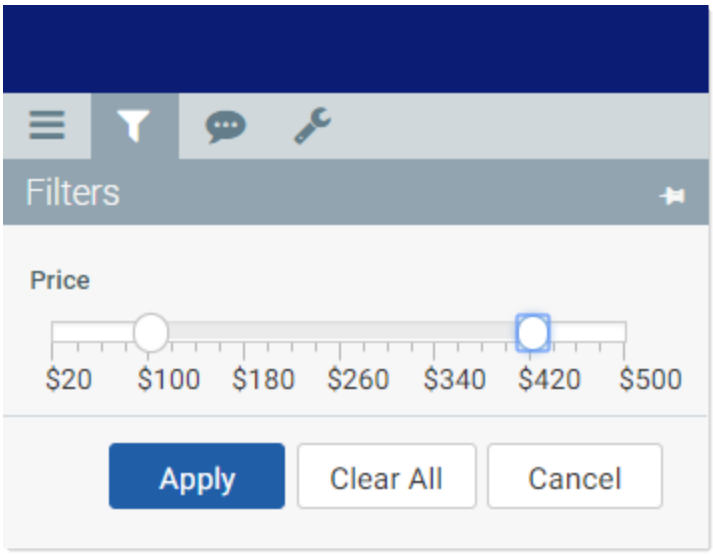
Axiom supports two types of variables that use slider handles. If you want users to be able to select a single value instead of a range, use a Slider refresh variable instead.

Variable behavior

The variable displays as a bar with two slider handles. The user can slide the handles along the bar to select a top value and a bottom value, within the overall defined range of values.



Desktop Client: Example Range Slider refresh variable



Web Client: Example Range Slider refresh variable

As a handle slides across the bar, the values show in a tooltip so that users can see the available values between the labeled tick marks (if applicable).

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a Range Slider variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define Range Slider variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

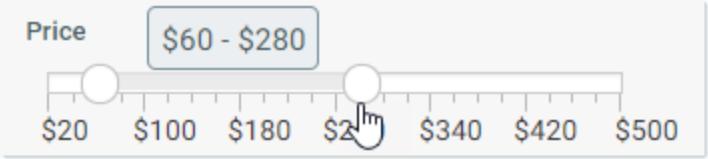
For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

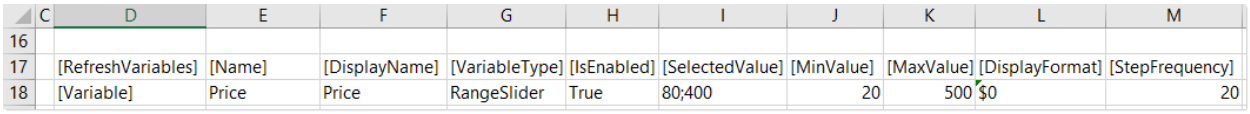
The following additional properties apply to Range Slider variable types:

Column Tag	Description
[MinValue]	The minimum value for the slider bar. This value defines the smallest number that can be selected on the slider. The left slider handle starts at the minimum value if no default value is specified.
[MaxValue]	The maximum value for the slider bar. This value defines the largest number that can be selected on the slider. The right slider handle starts at the maximum value if no default value is specified.
[StepFrequency]	Optional. The frequency of step values within the range. For example, if the range is 0 to 100 and you specify a step frequency of 10, then step values are available at 0, 10, 20, and so on. The slider handles snap to step values, so the step frequency determines which values the user can select within the range. The slider bar shows up to seven major tick marks with step values. All other steps are shown using minor tick marks, with the values visible in a tooltip as the user slides a handle.  *Example range slider tooltip* If the step frequency is not specified, then by default the range is divided evenly into 50 step values.
[DisplayFormat]	Optional. A display format for the values in the range, using an Excel number format string. Only very basic numeric formats are supported, such as to specify decimal places, currency symbols, and percentage signs. More advanced number formats, like colors and parentheses for negative numbers, are not supported. For example: • `0` for whole numbers • `0.0` for numbers with one decimal place • `0.0%` for percentages with one decimal place **IMPORTANT:** The number format must be entered as a string. You can precede the number with an apostrophe to cause it to be treated as text, such as `'0.0%`, or you can change the cell format to Text. If you enter just a number into the cell, then it will not be recognized as a number format string, even if the number is formatted the way you want.

The following properties do not apply to RangeSlider variables: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, TooltipColumn, PrimaryTable, LimitColumn, AutoQuoteString, MinDate, MaxDate.

Example data source

The following screenshot shows an example RangeSlider variable. The Selected Value field is populated to show what the values look like in the spreadsheet after the user selection.



	C	D	E	F	G	H	I	J	K	L	M
16											
17		[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsEnabled]	[SelectedValue]	[MinValue]	[MaxValue]	[DisplayFormat]	[StepFrequency]
18		[Variable]	Price	Price	RangeSlider	True	80;400	20	500	$0	20

RelatedColumnValue refresh variable

RelatedColumnValue refresh variables can be used to bring in additional column values based on the user's selection for a parent variable. This is a special variable type that does not display to the user—it is only used to return values that can then be referenced elsewhere in the file.

For example, imagine that you have a Grid variable for the user to select an account from Acct.Acct. The report will be filtered by that selected account, so you want to display both the account number and the account description in the report title. You can define a RelatedColumnValue variable that returns the associated value from Acct.Description for the selected account. You can then reference both values in the report title.

The RelatedColumnValue variable provides an alternative to using GetData functions to return these associated values. Avoiding use of unnecessary GetData functions can improve file performance.

The following variable types can be parent variables for a RelatedColumnValue variable:

- Grid variables
- ComboBox variables using a table column
- ComboBox variables using an Axiom query

The parent variable must use a key column of a reference table as its value column in order to bring in related column values.

Variable behavior

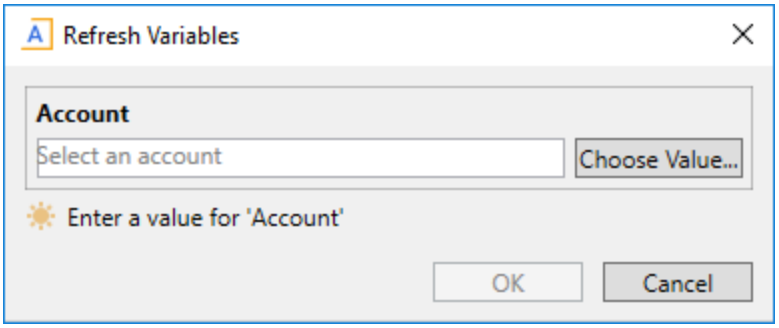
RelatedColumnValue variables do not display to users on refresh. Instead, they are automatically populated based on the user's selection for a parent variable. Once the user has completed the selections for the other refresh variables and those selections are written to the data source, the `[SelectedValue]` column for the RelatedColumnValue variable is also populated with the appropriate value from the specified column for the variable.

For example, imagine that you have the following configuration:

	A	B	C	D	E	F	G	H	I
1									
2		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[DependsOn]	[ColumnName]
3		[Variable]	Account	Grid	TRUE		TRUE		Acct.Acct
4		[Variable]	Description	RelatedColumnValue	TRUE			Account	Acct.Description

In this example, the RelatedColumnValue variable named Description depends on the Grid variable named Account. The Description variable is configured to return the value for Acct.Description based on the selected account for the Account variable.

When the **Refresh Variables** dialog is presented to the user (or the Web Client filter panel), only the Account variable displays:



Once the user selects an account and clicks **OK**, the selected account is placed in the `[SelectedValue]` column for the Account variable, and the associated description from Acct.Description is placed in the `[SelectedValue]` column for the Description variable. In the following example, the user selected account 5000, and the description of that account is "Travel."

	A	B	C	D	E	F	G	H	I
1									
2		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[DependsOn]	[ColumnName]
3		[Variable]	Account	Grid	TRUE	5000	TRUE		Acct.Acct
4		[Variable]	Description	RelatedColumnValue	TRUE	Travel		Account	Acct.Description

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a RelatedColumnValue variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define RelatedColumnValue variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

NOTE: The `[DisplayName]`, `[GroupName]`, and `[CollapseOnOpen]` properties do not apply to RelatedColumnValue variables because these variables do not display to users.

Variable-specific properties

The following additional properties apply to RelatedColumnValue variable types:

Column Tag	Description
`[ColumnName]`	Specifies the related column from which to look up a value, based on the selected value for the parent variable. Enter a fully-qualified Table.Column name such as `Acct.Description`. The column for the parent variable must be a key column of a reference table or a validated column. • If the parent variable column is the key column of a reference table, then the related column can be any column from the same table as the parent variable, or from a table that the parent variable has a lookup relationship with. For example, if the parent variable column is `Dept.Dept`, then the related column can be any other column in that table such as `Dept.Description` or `Dept.Region`. It can also be `Dept.Region.RegionType`, assuming that `Dept.Region` looks up to `Region.Region`. • If the parent variable column is a validated column, then the related column can be any column from the lookup table. For example, if the parent variable column is `Dept.Region`, then the related column could be `Dept.Region.RegionType`. If the parent variable is a ComboBox variable that uses an Axiom query, then the related column must be present in the field definition of that query.
`[AutoQuoteString]`	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. **NOTE:** This option only applies if the related column is a string column.

The following properties do not apply to RelatedColumnValue variables: PlaceHolderText, ListChoices, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Slider refresh variable

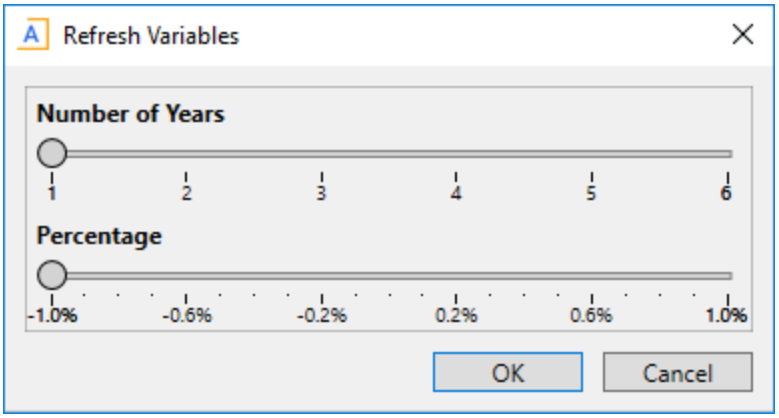
Slider refresh variables prompt users to select a value along a defined range of values, using a slider handle. The value is written back to the data source, where it can be used to impact data queries or other aspects of a report.

For example, you can use the slider variable to prompt users to select a number of years to show in the report. The value can then be used to impact the number of columns that are included in the query (or just which years are currently visible).

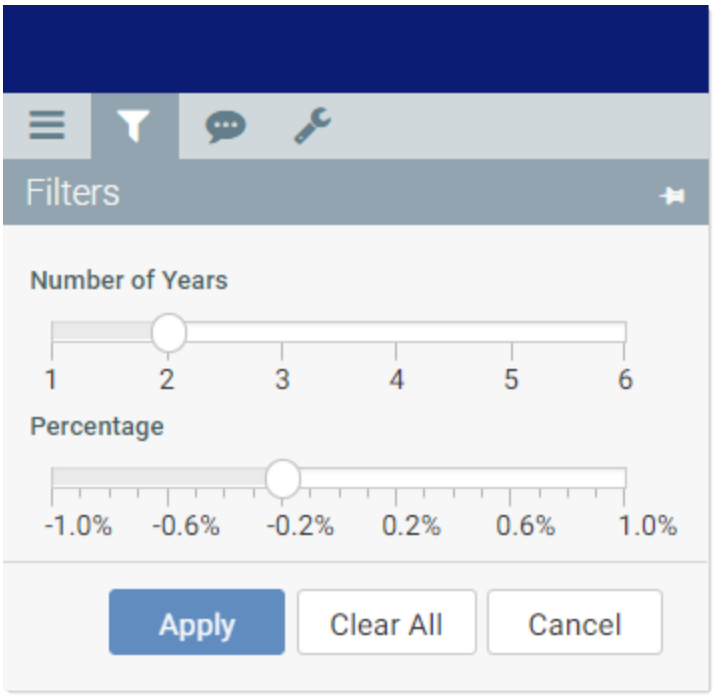
Axiom supports two types of variables that use slider handles. If you want users to be able to select a range of values instead of just a single value, use a RangeSlider refresh variable instead.

Variable behavior

The variable displays as a bar with a slider handle. Users can slide the handle along the bar to select a value within the defined range.



Desktop Client: Example Slider refresh variable



Web Client: Example Slider refresh variable

As the handle slides across the bar, the values show in a tooltip so that users can see the available values between the labeled tick marks (if applicable).

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a Slider variable. Some data source columns do not apply in this case and are not discussed here. If these inapplicable columns are present in the data source, they should be left blank on rows that define Slider variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

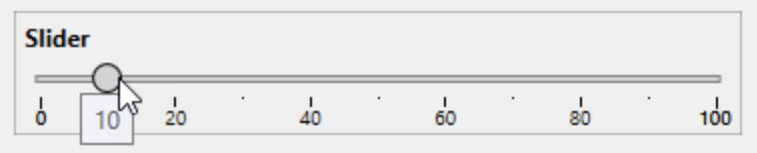
For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

The following additional properties apply to Slider variable types:

Column Tag	Description
[MinValue]	The minimum value for the slider bar. This value defines the smallest number that can be selected on the slider. The slider handle starts at the minimum value if no default value is specified.
[MaxValue]	The maximum value for the slider bar. This value defines the largest number that can be selected on the slider.
[StepFrequency]	Optional. The frequency of step values within the range. For example, if the range is 0 to 100 and you specify a step frequency of 10, then step values are available at 0, 10, 20, and so on. The slider handle snaps to step values, so the step frequency determines which values the user can select within the range. The slider bar shows up to seven major tick marks with step values. All other steps are shown using minor tick marks, with the value visible in a tooltip as the user slides the handle.  *Example slider tooltip* If the step frequency is not specified, then by default the range is divided evenly into 50 step values.
[DisplayFormat]	Optional. A display format for the values in the range, using an Excel number format string. Only very basic numeric formats are supported, such as to specify decimal places, currency symbols, and percentage signs. More advanced number formats, like colors and parentheses for negative numbers, are not supported. For example: • `0` for whole numbers • `0.0` for numbers with one decimal place • `0.0%` for percentages with one decimal place **IMPORTANT:** The number format must be entered as a string. You can precede the number with an apostrophe to cause it to be treated as text, such as `'0.0%`, or you can change the cell format to Text. If you enter just a number into the cell, then it will not be recognized as a number format string, even if the number is formatted the way you want.

The following properties do not apply to Slider variables: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, Hierarchies, UseAsQuickFilter, TooltipColumn, PrimaryTable, LimitColumn, AutoQuoteString, MinDate, MaxDate.

Example data source

The following screenshot shows example Slider variables:

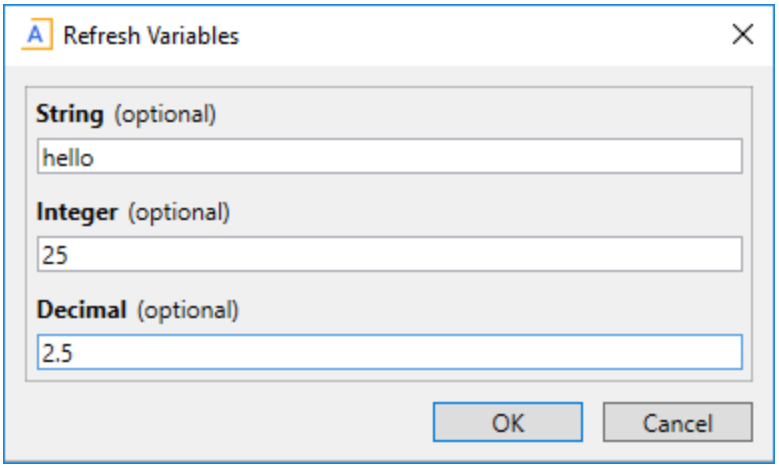
	C	D	E	F	G	H	I	J	K	L	M
13		[RefreshVariables]	[Name]	[DisplayName]	[VariableType]	[IsEnabled]	[SelectedValue]	[MinValue]	[MaxValue]	[DisplayFormat]	[StepFrequency]
14		[Variable]	SliderNum	Number of Years	Slider	True		1	6	0	1
15		[Variable]	SliderPercent	Percentage	Slider	True		-1.0%	1.0%	0.0%	0.10%

String, Integer, and Decimal refresh variables

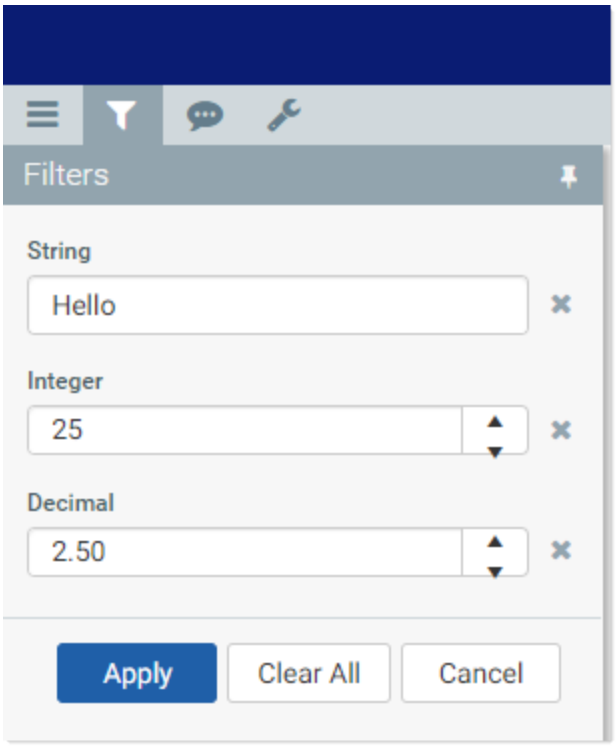
String, Integer, or Decimal refresh variables prompt users to make a "free input" of the designated type. For example, if the variable is an Integer variable, then the user can enter any whole number. However, the user cannot enter any decimal numbers or any text.

▶ Variable behavior

The variable displays as a text box where the user can type in their desired value for the variable. In the Web Client only, Integer and Decimal variables also have up and down arrows at the side of the text box to increase or decrease the current input.



Desktop Client: Example free-input refresh variables



Web Client: Example free-input refresh variables

Invalid entries—for example, attempting to enter a decimal value into an Integer variable—are treated as follows:

- In the Desktop Client, the user can enter any value into the text boxes. However, if the value is invalid, then a validation message displays at the bottom of the dialog. The **OK** button is disabled until the user enters a valid value.
- In the Web Client, invalid values are prevented at the point of entry, so no validation messages display.

NOTE: In the Web Client, decimal values for refresh variables are limited to hundredths (two places to the right of the decimal point).

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a String, Integer, or Decimal variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define free-input variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

Column Tag	Description
`[AutoQuoteString]`	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. This option applies to String variables but not to Integer or Decimal variables.

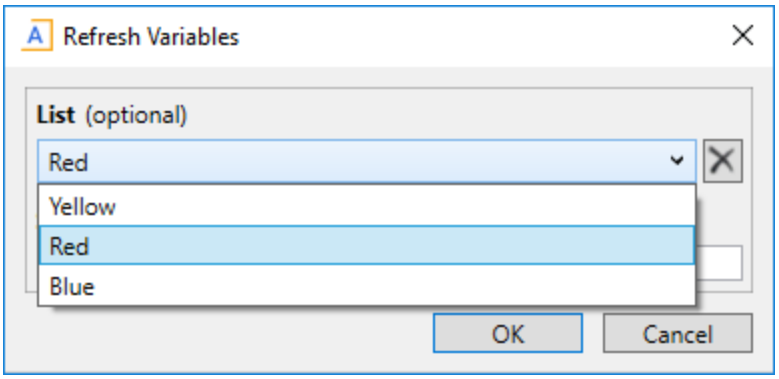
The following properties do not apply to String, Integer, or Decimal variables: PlaceHolderText, ListChoices, ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

StringList refresh variable

StringList refresh variables prompt users to make a selection from a predefined list. This list is manually defined within the current file—either in the RefreshVariables data source itself, or in a range of cells that are then referenced in the data source.

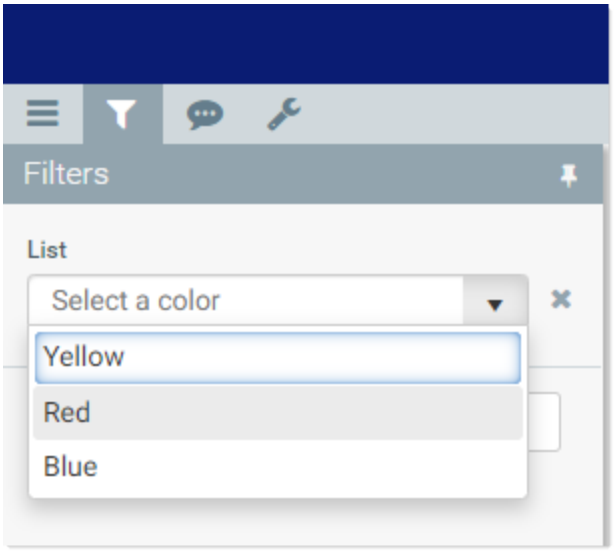
▶ Variable behavior

In the Desktop Client, the variable displays as a drop-down list. The user must use the drop-down to select a value.



Desktop Client: Example StringList variable

In the Web Client, the variable displays as a drop-down list with a searchable entry box. The user can scroll the list and select the value directly, or type into the box to find a particular value.



Web Client: Example StringList variable

Variable properties

This section explains how to complete a variable row in the RefreshVariables data source when defining a StringList variable. Some data source columns do not apply in this case and are not discussed here. If any inapplicable columns are present in the data source, they should be left blank on rows that define StringList variable types. If you are using the Data Source Assistant to complete the variable properties, then only the applicable columns will be shown in the task pane.

For more information on the RefreshVariables data source in general, see Defining refresh variables.

General variable properties

All refresh variables use a common set of general properties such as the variable name, display name, and whether the variable is enabled or required. For more information on these properties, see General refresh variable properties.

Variable-specific properties

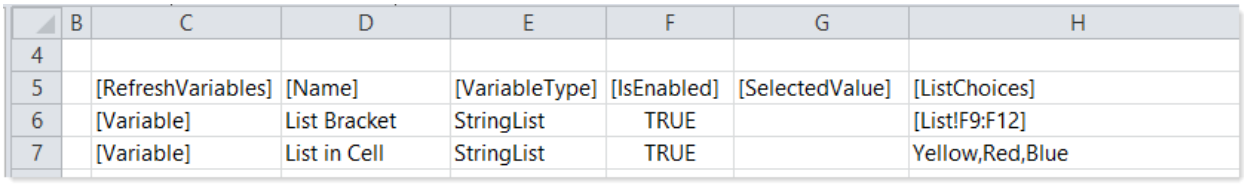
The following additional properties apply to StringList variable types:

Column Tag	Description
`[ListChoices]`	Defines the list of values for the variable. Enter one of the following: • A comma-separated list of values to display in the list. For example: `Yellow, Red, Blue`. (Alternatively, semicolons can be used as the delimiter.) **NOTE:** This list can contain cell references in brackets. The value for the list will be read from the designated cell. For example, you can enter: `Yellow, [List!B5], Blue`. If List!B5 contains the value Red, then the list will be: Yellow, Red, Blue. If List!B5 contains a cell range, then the values within that range will be added to the list. • A cell range, in brackets. For example: `[List!B10:B20]`. The list of values will be read from this range. When using a cell range, the range must be one dimensional—meaning either one column wide and several rows tall, or several columns wide and one row tall. If the range is a block, it is ignored.
`[PlaceholderText]`	Optional. Specifies placeholder text to display within the variable box until a value is selected. If blank, then the default text "Select..." is used. For StringList variables, this property is only supported for use in the Web Client. In the Desktop Client, the variable box is blank if no value has been selected.
`[AutoQuoteString]`	Optional. Specifies whether the string value is placed in single quotation marks when it is written to the `[SelectedValue]` column (True/False). If omitted or blank, the default setting is False, which means the string value is not quoted. This option is intended to make it easier to create filters based on the selected value, when the selected value is a string and therefore must be wrapped in single quotation marks. For example: `Dept.VP='Smith'`. **NOTE:** All values in the list are considered strings and will be quoted if this option is enabled, even if the list values are actually numbers.

The following properties do not apply to StringList variables: ColumnName, AdditionalColumns, ColumnFilter, AllowMultiSelect, DataSourceName, DisplayFormat, Hierarchies, UseAsQuickFilter, MinDate, MaxDate, TooltipColumn, PrimaryTable, LimitColumn, MinValue, MaxValue, StepFrequency.

Example data source

The following screenshot shows a couple of StringList refresh variable examples. The first example reads the list from a cell range, while the second example uses a comma-separated list.



	B	C	D	E	F	G	H
4							
5		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[ListChoices]
6		[Variable]	List Bracket	StringList	TRUE		[List!F9:F12]
7		[Variable]	List in Cell	StringList	TRUE		Yellow,Red,Blue

Using the Data Source Assistant to add or edit refresh variables

Once the RefreshVariables data source has been placed in a sheet, you can use the Data Source Assistant to add or edit refresh variables. For general information on the Data Source Assistant and its availability, see Using the Data Source Assistant.

▶ Adding a new variable to the data source

To add a new variable, place your cursor on a blank row in the data source. In the **Selection Editor** section, select a **Variable Type**. This will add a `[Variable]` tag to the current row, and populate the `[VariableType]` field with the selected type. You can then use the Selection Editor to complete the remaining settings for the variable.

Alternatively, you can also add a new variable row using the **Edit Tags** section:

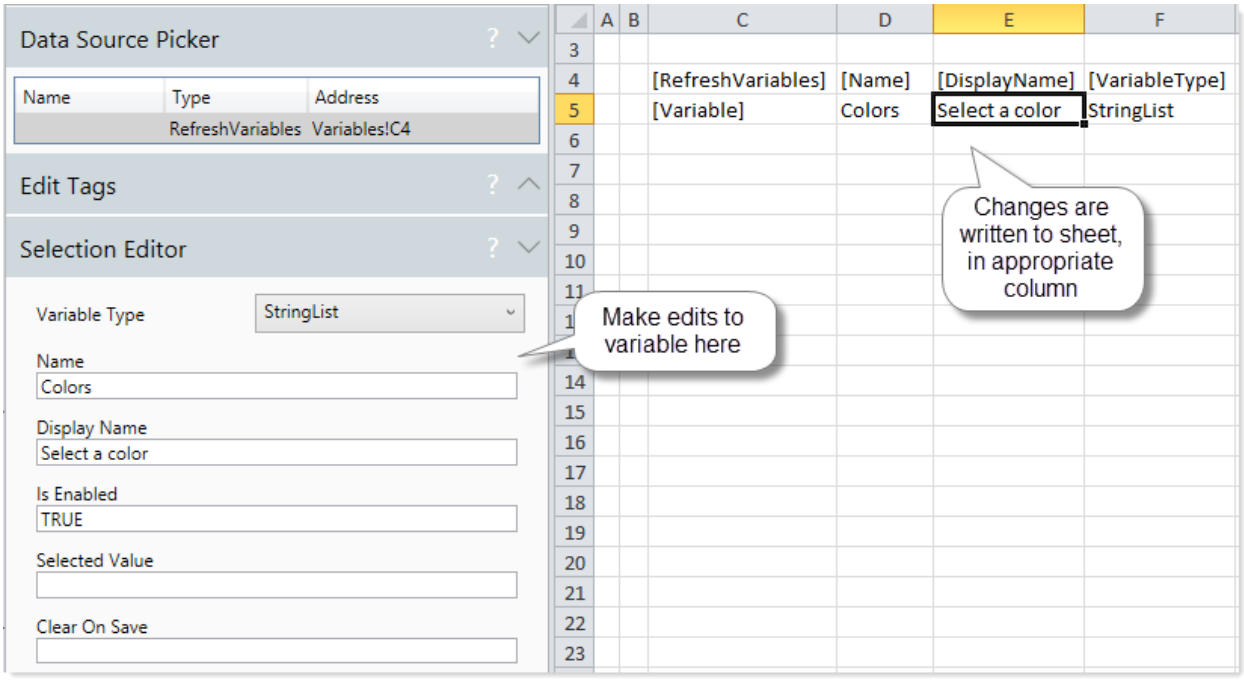
- Place your cursor in a blank row within the data source. Under **Row Tags**, click **Update** for Variable. This adds a `[Variable]` tag to the current row.
- Place your cursor in a populated row within the data source. Under **Row Tags**, click **Insert** for Variable . This inserts a new row above the current row, and places a `[Variable]` tag in the new row.

Once you have updated or inserted a variable row, you can go back to the Selection Editor section and select a **Variable Type** for the row. The Selection Editor will then populate with the applicable properties for that row, which you can edit as needed.

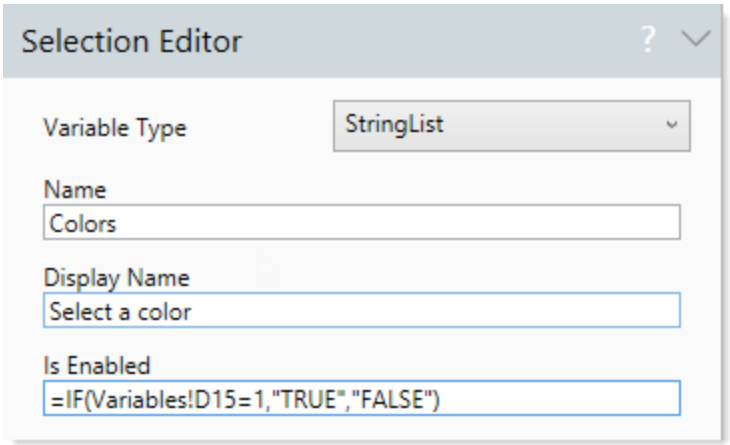
NOTE: If the RefreshVariables data source does not already contain all of the columns used by the selected variable type, these columns will be automatically added to the end of the data source.

Editing refresh variables

To edit an existing variable, place your cursor within the row, in the data source—either on the `[Variables]` tag or in one of the property columns. The properties for the variable display in the **Selection Editor** section of the Data Source Assistant, filtered by variable type. You can modify any of these properties and the changes are written back to the sheet, in the appropriate columns.



If a particular property uses a formula, then that formula displays when you place your cursor in the property box. You can then edit that formula as needed.



The Selection Editor only displays variable properties where the corresponding column already exists in the sheet. For example, the property **Clear On Save** only displays if the column tag `[ClearSelectedValueOnSave]` exists in the sheet. If necessary, you can also use the Data Source Assistant to add missing tags to the data source. See Adding property columns.

Adding property columns

When you use the right-click wizard to insert the RefreshVariables data source, all column tags are placed in the sheet by default. Therefore in most cases you should not need to add columns, unless:

- Columns were deleted from the data source after it was inserted into the sheet. You might do this to clean up the sheet if you are not currently using a particular property.
- Your organization has upgraded to a new Axiom version that supports new column tags. In this case you must add the column tags to any existing data source if you want to use them.

To add new column tags to a data source, use the **Edit Tags** section to do either of the following:

- Place your cursor in a blank column in the data source. Under **Column Tags**, locate the tag that you want to add and then click **Update**. This adds the tag to the current column.
- Place your cursor in a populated column within the data source. Under **Column Tags**, locate the tag that you want to add and then click **Insert**. This inserts a new column to the left of the current column, and places the tag in the column.

Column tags are only listed if the data source does not already contain that tag. If your data source already contains all tags, then no column tags will be listed in the Edit Tags section.

Once you have updated or inserted a new column tag, you can populate the column manually or you can use the Selection Editor section to update each variable row for the new property.

Using dependent refresh variables

You can make one refresh variable dependent on the selected value of another refresh variable. If the dependent variable is configured with dynamic settings—such as using formulas to determine the list or column or filter—then the dependent variable can change based on the selected value of the "parent" variable. You can also determine whether the dependent variable displays at all, by making the `[IsEnabled]` property for the variable dynamic.

Dependent variables work as follows:

- When the variables are presented to the user in the refresh dialog or filter panel, the parent variable is editable and the dependent variable is not. The dependent variable may be visible but grayed out, or it may not display at all (depending on whether the dependent variable is enabled initially).
- When the user specifies a value for the parent variable, that value is placed in the selected value cell for that variable, and the file is calculated.

- The settings for the dependent variable are re-read, and the refresh dialog or filter panel is updated for any changes. The user can now select a value for the dependent refresh variable, assuming the dependent variable is enabled.

This behavior applies in both environments—the **Refresh Variables** dialog in the Desktop Client and the **Filter** panel in the Web Client.

Configuring a variable as dependent

If you want a variable to be dependent on another variable, set the `[DependsOn]` property for the dependent variable to the name of the "parent" variable. For example, if the Account variable is dependent on the value selected for the Category variable, then enter `Category` in the `[DependsOn]` property for the Account variable.

Once a variable has been designated as dependent on another variable, then you can use formulas in its variable settings to dynamically change the variable based on the parent variable selection. For example, you could use formulas to:

- Specify whether the variable is enabled or not (see the following visibility discussion for more information)
- Change the list choices for the variable
- Change the Table.Column for the variable
- Change the filter applied to the column

NOTE: The variable name cannot be dynamic; it must remain static because it is used to identify the variable. If you need the name of the dependent variable to change based on the selection of the parent variable, then you must make the display name dynamic instead of the name.

In all of these cases, the formula should look to the selected value of the parent variable and change in response. For example, users could specify a numeric value for the parent variable, and then the dependent variable could be enabled or not depending on whether the numeric value was greater than a certain amount. Or users could select column names from the parent variable (such as Dept, VP, or Region), and then the column value for the dependent variable could change based on the parent variable selection.

The calculation and update of the refresh dialog or filter panel only occurs if a variable is flagged as dependent using the `[DependsOn]` property, and only dependent variables are updated. If you use formulas in variable settings but the variable is not flagged as dependent, then the variable will not update once the refresh dialog or filter panel has been presented to the user.

Visibility of dependent variables in the refresh dialog or filter panel

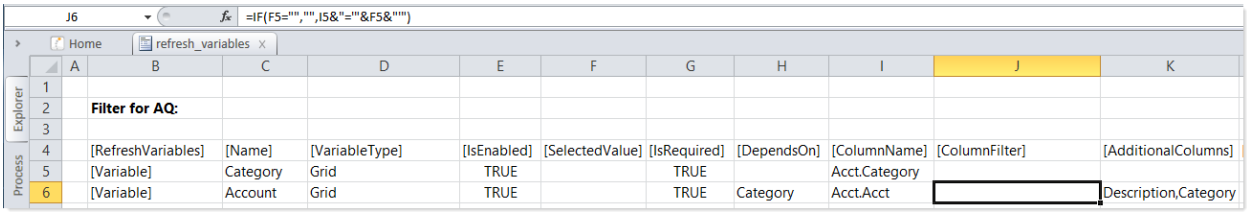
When setting up dependent variables, you should consider whether you want the dependent variable to be visible in the refresh dialog or filter panel from the start, or whether it should be hidden initially. Dependent variables behave as follows, based on the value of the `[IsEnabled]` property:

- If the dependent variable is enabled, then the dependent variable displays in the refresh dialog or filter panel, but it is grayed out and unavailable to the user. Once the user selects a value for the parent variable, the dependent variable becomes available for use.
- If the dependent variable is not enabled, then the dependent variable is omitted from the refresh dialog or filter panel. Once the user selects a value for the parent variable, the dependent variable may or may not become visible in the refresh dialog or filter panel, depending on the formula that was used in the enabled setting.

  For example, the formula might be set up so that once any value is set for the parent variable, the dependent variable becomes enabled. Or the formula might be set up so that the dependent variable only becomes enabled if the parent variable value meets a certain criteria.

Dependent variable example

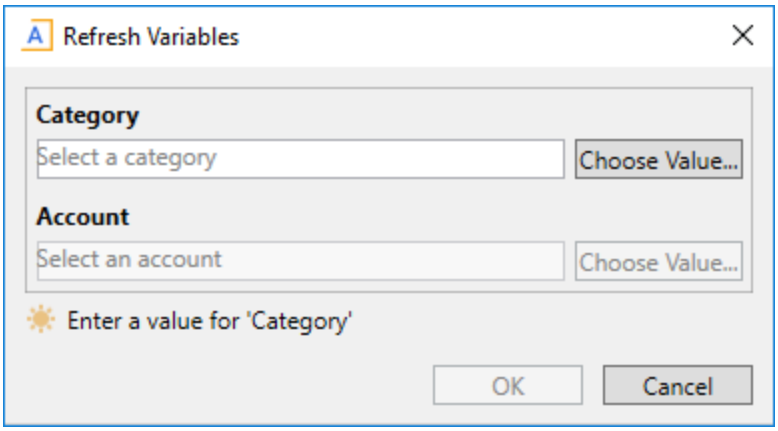
A common dependency example is to use one variable to define a filter for a second variable. In this example, the Category variable (the parent variable) is used to select an account category. The Account variable (the dependent variable) then presents a list of accounts, filtered by the selected category.



	A	B	C	D	E	F	G	H	I	J	K
1											
2		**Filter for AQ:**									
3											
4		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[DependsOn]	[ColumnName]	[ColumnFilter]	[AdditionalColumns]
5		[Variable]	Category	Grid	TRUE		TRUE		Acct.Category		
6		[Variable]	Account	Grid	TRUE		TRUE	Category	Acct.Acct		Description,Category

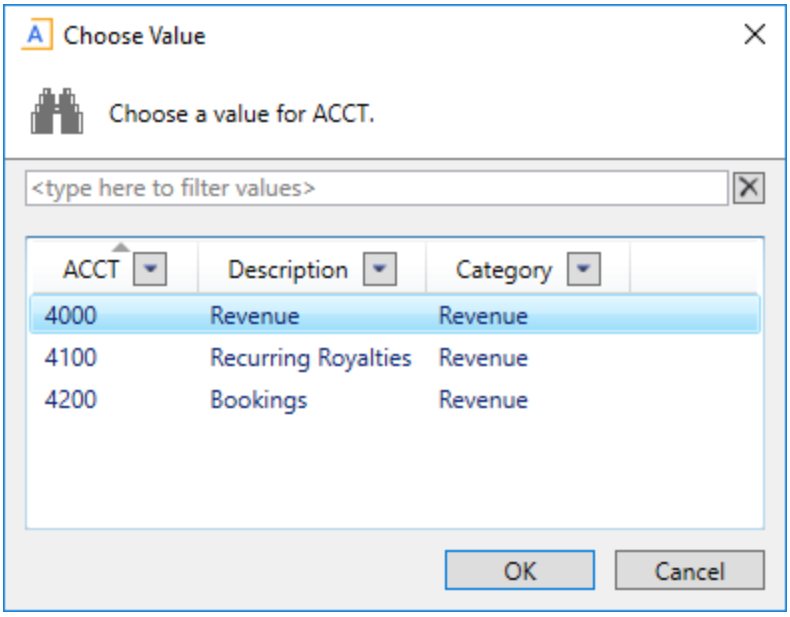
To filter the Account variable by the value selected for the Category variable, the `[ColumnFilter]` cell for the Account variable contains a formula that looks to the `[SelectedValue]` cell for the Category variable. Right now the filter cell is blank because no category has yet been selected.

When the user refreshes the file in the Desktop Client, the refresh dialog appears as follows:



The dependent variable, Account, is initially grayed out, and the user must define the Category variable first. Once the user selects a category, that category is placed in the sheet (in the `[SelectedValue]` cell in the data source) and then the file is calculated. This process occurs in the background and is transparent to the user.

The Account variable is now available to be selected. When the user clicks on Choose Value, the list of accounts is filtered based on category (for example, Revenue), like so:



The user's account selection is placed in the sheet, and the file is refreshed.

The following is an example of how the variable values and filter creation could look in the worksheet once all variables have been selected. In this example, we are taking the final account selection and creating a filter to apply to an Axiom query.

	A	B	C	D	E	F	G	H	I	J	K
1											
2		**Filter for AQ:**	Acct.Acct=4000								
3											
4		[RefreshVariables]	[Name]	[VariableType]	[IsEnabled]	[SelectedValue]	[IsRequired]	[DependsOn]	[ColumnName]	[ColumnFilter]	[AdditionalColumns]
5		[Variable]	Category	Grid	TRUE	Revenue	TRUE		Acct.Category		
6		[Variable]	Account	Grid	TRUE	4000	TRUE	Category	Acct.Acct	Acct.Category='Revenue'	Description,Category

Design considerations

When configuring dependent variables, you should consider the following:

- Within the RefreshVariables data source, the parent variable should be placed before the dependent variable, because variables are presented to users in the order they are found in the data source. If the parent variable is located after the dependent variable, then the variable selections may not make sense to the user as they are presented in the refresh dialog or filter panel.
- The parent variable can be required or not as appropriate. If the parent variable is not required and no selection is made for that variable, then the dependent variable will never become active and the user will not be able to make a selection for that variable. If you need the user to make a selection for the dependent variable, then both the parent variable and dependent variable should be required.

If you want to use a CheckBox variable as a parent variable, so that the dependent variable changes based on whether the check box is selected or not, then you must do one of the following:

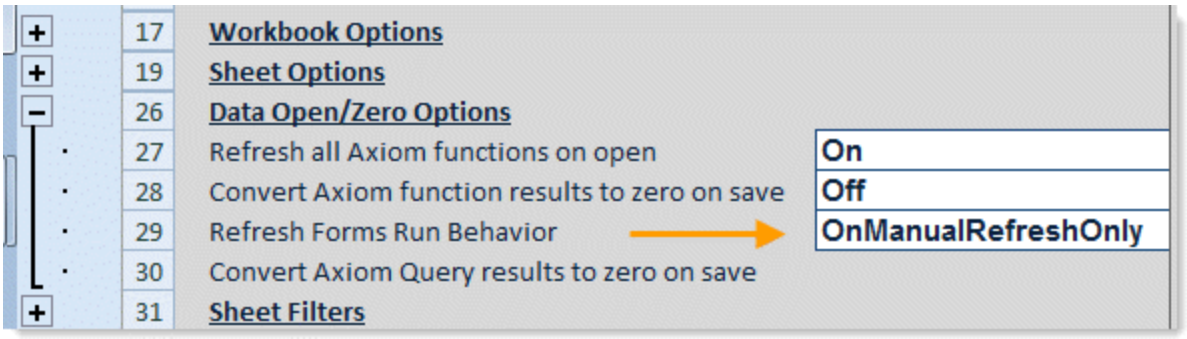
- Configure the CheckBox variable as required, and define a default value for the variable in the `[SelectedValue]` field. This is to ensure that the variable always has a selected value of True or False.
- Otherwise, make sure that your formulas are set up to account for the blank case and treat it the same way as an entry of False.

Determining when refresh variables display to users

You can configure the run behavior of refresh variables, meaning you can determine which refresh actions trigger the display of the refresh dialog.

NOTE: This behavior option only applies when using refresh variables with spreadsheet Axiom files. In Axiom forms, refresh variables are always available in the Filters panel.

This setting is defined on the Control Sheet, in the **Data Open/Zero Options** section.



The **Refresh Forms Run Behavior** setting has the following options:

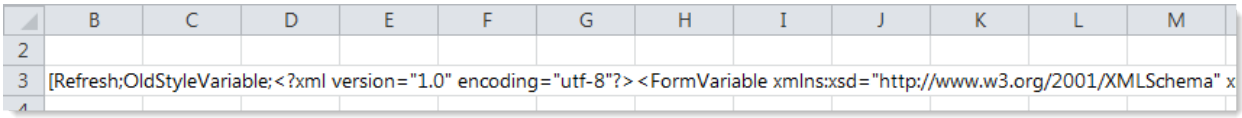
- **Off**: The refresh dialog is disabled and does not display when the file is refreshed.
- **OnManualRefreshOnly** (default): The refresh dialog only displays when the file is manually refreshed (by clicking **Refresh**).
- **OnOpenOnly**: The refresh dialog only displays when the file is refreshed 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**: The refresh dialog displays when the file is refreshed on open, and when the file is manually refreshed.

The refresh dialog is always disabled 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.

This setting also applies when using an Axiom form as a refresh dialog.

Converting XML refresh variables to a RefreshVariables data source

In older versions, Axiom supported a different way to define refresh variables. This older method used an XML tag to define the variable settings. The following screenshot shows an example of how an XML-style variable appears within a file:

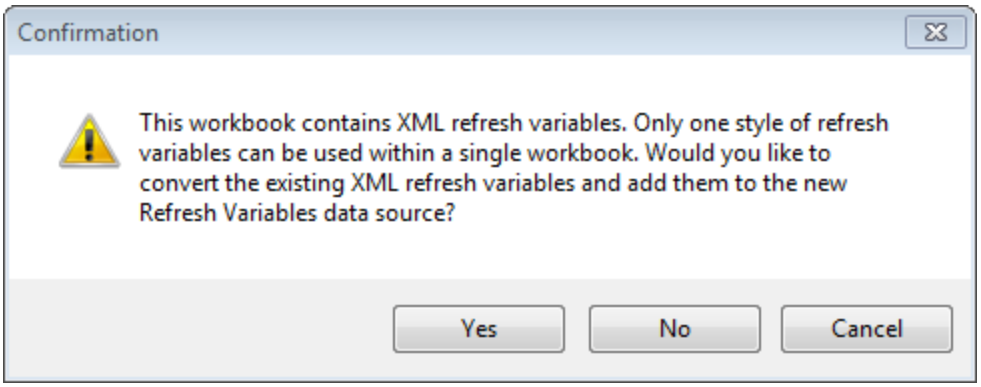


These XML-style variables have been deprecated in favor of using the RefreshVariables data source. Although existing files with XML-style variables will still work, it is strongly recommended to convert these older-style variables to the new RefreshVariables data source. The RefreshVariables data source provides greater flexibility to create dynamic refresh variables, and will be the focus of any new refresh variable enhancements moving forward.

Axiom provides a tool to easily convert your existing variables. This tool will create a RefreshVariables data source, then locate all existing variables in the file and convert the XML settings to `[Variable]` rows in the data source.

To convert existing refresh variables:

1. Identify the sheet in the file where you want to place the new RefreshVariables data source. Right-click the cell where you want the data source to start (this is the cell where the primary `[RefreshVariables]` tag will be placed), and then click **Axiom Wizards > Insert Refresh Variable Data Source**.
2. If the file contains any XML-style refresh variables, Axiom asks if you want to convert these variables. Click **Yes**.



Axiom inserts the data source tags and adds the existing variables to the data source (assuming there is enough room at the selected location—if there is not enough room, a message box will inform you how much room is necessary so that you can select a new location).

Alternatively, you can right-click an existing XML-style variable in the workbook and select **Axiom Wizards > Convert XML Refresh Variables to Data Source**. This will place the data source at the existing location of the variable, and will also convert all XML-style variables in the workbook and add them to the data source. However, in many cases there is not enough space at existing variable locations to add a full data source, so it is typically cleaner to start at a new location.

Note the following about the refresh variable conversion:

- **Target Cell to [SelectedValue]**: The conversion from using a Target Cell to using the `[SelectedValue]` column works as follows:
  - If the original target cell contains a value, this value is copied to the selected value cell for the new variable.
  - A formula is placed in the original target cell that points to the selected value cell for the new variable.

For example, imagine that the old refresh variable was defined on the sheet Info, and the variable's target cell was B1. The Info!B1 cell contained the text "Corporate". After the conversion, the Info!B1 cell now contains a formula something like `=Variables!G15` (assuming that the new data source was inserted on the Variables sheet, and the `[SelectedValue]` column is in column G). The Variables!G15 cell contains the text "Corporate".

This is done so that any existing formulas in the workbook that are pointing to the old target cell will still work. You can go forward with using the file like this, or you can adjust the existing formulas so that they point directly to the selected value cell instead of the old target cell.

- **Bracketed Cell References**: The old XML syntax supported bracketed cell references to read information from the workbook. This was necessary because formulas could not be used in the old refresh variable properties. Bracketed cell references in the old variable are converted as follows:
    - Bracketed cell references in the Name property are converted to formulas in the `[Name]` column.
    - Bracketed cell references in the Column Filter property (for Column Value variables) are converted to formulas in the `[ColumnFilter]` column.
    - Bracketed cell references in the List Items property are moved to the `[ListChoices]` column as is. This is because the cell references may be combined with regular text items, and because the old List Items property supported use of ranges as well as single cell references. These bracketed cell references will continue to work in the new RefreshVariables data source. However, if it makes sense to change any of these references to use formulas instead, feel free to do so after conversion.
- **Deleting Old Variables**: The conversion process does not delete the old XML-style variables from the workbook. (Except in the case where you convert at an existing variable location, in which case that single existing variable is overwritten by the data source.) Once you have reviewed the conversion and confirmed that the file is working as you expect, you should delete these old variables from the workbook.

8

Saving Data to the Database

You can save data from an Axiom file to the database. This can be used for various purposes, such as:

- Saving planning data from plan files
- Saving driver data from driver files
- Saving allocations or other calculated data from a report utility
- Saving changes to table properties or columns

Only Axiom managed files can save data to the database.

Overview of save types

Any managed Axiom file can be configured to save data to the database. Save settings are configured on a per sheet basis, on the Control Sheet.

Axiom supports several "save types" that can be used for different purposes:

- **Save Type 1**: This is the most flexible save type, and is the one most likely to be used in templates/plan files and report files. To set up Save Type 1, you use tags within the sheet to establish a save-to-database control row and control column, and to determine what data is saved or deleted. The destination database table and columns must already exist in the database.
- **Save Type 3**: This is a structured save type, used to create *document reference tables* for drivers or other data. Unlike Save Type 1, Save Type 3 creates the table in the database (if it does not already exist). The table is based on the sheet structure, and all edits are managed within the sheet. When a save is performed, the table is edited to match the current structure in the spreadsheet. If a row or column is deleted in the sheet, the corresponding row or column is removed from the table.
- **Save Type 4**: This save type provides an alternate method for performing certain system administration tasks. For example, instead of creating or editing column aliases within the table editor, you can modify them from within a spreadsheet interface using Save Type 4. This save type is set up in a similar manner as Save Type 1, with a save-to-database control row and control column, but using different tags.

Axiom supports another save type, Save Type 2, which is the method used by the **Open Table in Spreadsheet** feature to view and edit data and reference tables within a spreadsheet interface. Save Type 2 is a highly structured save type that is only used for this system-managed editing, and cannot be used by other files in Axiom.

Security settings and saving data to the database

If a file is configured to save data to the database, a user's security settings impact whether that user can perform the data save:

- For report files, driver files, and file group utilities, the option **Allow Save Data** must be enabled for the folder or for the specific file, on the **Files** tab of Security.
- For plan files, users must also have the **Allow Save Data** permission to the plan file. The permission may be explicitly granted to the user in security, or the user may have their permissions "elevated" due to being a step owner of the plan file in a plan file process.

Assuming the user has rights to perform the save within the file, the user must also have write rights to the target table, as defined on the Tables tab of **Security**.

Administrators do not need to be explicitly granted any permissions; they automatically have the necessary permissions to save data.

Comparison of save types

The following table summarizes the defining characteristics of each save type:

Save Type	Location of Database Codes	Target Table and Columns	Behavior to Delete and/or Zero Data
1	Anywhere; depends on placement of tags in the sheet. Typically used in templates/plan files and reports.	Target table and columns must already exist in the database. Can save to data tables or reference tables.	If **Zero on Save enabled** is **On**, data from the prior save is zeroed before performing the new save (based on the document ID or a custom save tag). Rows of data can be deleted by flagging the row for deletion instead of save.
3	Column A, and rows 1-3. Typically used in driver files.	The table and columns are created based on the sheet structure. Creates a document reference table.	The table and columns are re-created to match the spreadsheet data. If a row or column is deleted or zeroed in the spreadsheet, it is deleted or zeroed in the table.

Save Type	Location of Database Codes	Target Table and Columns	Behavior to Delete and/or Zero Data
4	Anywhere; depends on placement of tags in the sheet. Typically used in reports.	Modifies certain table structure settings and system settings.	Depends on the operation being performed. Zeroing does not apply to this process, but in some cases records can be deleted or restored to default values.

Note that blank rows and columns are permitted when using any save type. Blank rows and columns are skipped during the save process.

Save-to-database processing order

Each sheet in a file can have at least one enabled save type. If Save Type 3 is enabled for a sheet, then no other save types can be enabled for that sheet. Save Type 1 and Save Type 4 can both be enabled on the same sheet.

Sheets are processed in the order they are listed on the Control Sheet, from left to right. If both Save Type 1 and Save Type 4 are enabled for the same sheet, then Save Type 1 is processed before Save Type 4.

If Save Type 1 or Save Type 4 are enabled for a sheet, then that sheet can contain multiple save-to-database processes (Save2DB or SaveStructure2DB). Generally speaking, these tags are processed in the order they are found in the sheet, moving from left to right and top to bottom. It is also possible to specify an order for Save Type 1 tags in a sheet.

Using Save Type 1

Save Type 1 is the most flexible save type for saving data from an Axiom file to the database. You can use Save Type 1 in any managed Axiom file, but it is most typically used in templates/plan files and reports. Driver files typically use Save Type 3, but can optionally use Save Type 1 instead.

Save Type 1 is configured on a per sheet basis. Each sheet can have multiple save processes.

Save Type 1 does not create database tables or columns. The destination table and columns must already exist in Axiom. You can zero data by using Save Type 1, and you can delete rows of data, but you cannot delete columns.

NOTE: Save Type 1 cannot be used to save data to document reference tables, or to tables using the Large Table index scheme. Document reference tables can only be updated using Save Type 3. Tables using the Large Table index scheme can only be updated using an import utility or Copy Table Data.

To set up Save Type 1 for a sheet:

1. On the Control Sheet, configure the following settings:
   - If the sheet is not already set up on the Control Sheet, enter the sheet name into one of the definition columns.
   - In the **Save to Database Setup** section, set **Save Type 1 Enabled** to **On**.
   - In most cases, you will want to set **Zero on save enabled** to **On**. If the table does not already have a column named **SaveTagDocID** to hold the save tags, you must create it.

   In some cases, you may need to use advanced save options such as defining a custom zero tag and custom zero tag column. For more information on when these advanced options may be needed, see How Save Type 1 works.

   NOTE: The save-to-database settings are protected on the Control Sheet. You must unprotect the sheet using **Advanced > Protect > Worksheet** before you can configure a save-to-database process.

2. In the sheet, define the save-to-database control row and control column by placing the tag `[Save2DB;TableName]` in any cell. *TableName* is the destination table for the save-to-database process.

   This is the minimum required tag. The Save2DB tag supports additional optional parameters that may be used as needed.

3. In the save-to-database control row, for each column of data that you want to save to the database, enter the name of the target column in the destination table. For example, `ACCT` or `NYB1`.

   The control row must contain entries for all key columns and alternate key columns in the destination table. If you are saving new records to the destination table, all validated columns must be included as well (unless those columns have valid default values set in the column properties). All entries in the control row must belong to the same table.

4. In the save-to-database control column, for each row of data that you want to save to the database, enter the tag `[Save]`. For each row that you want to delete from the database, enter the keyword `[Delete]`.

5. If desired, set up a `[SaveError]` column to perform custom save validation on rows to be saved to the database. For more information, see Using custom save validation.

For more information on placing save-to-database tags in a sheet, including using advanced options such as a custom save tag or a custom zero tag, see Placing save-to-database tags in a sheet (Save Type 1).

Placing save-to-database tags in a sheet (Save Type 1)

Save Type 1 depends on the placement of save-to-database tags within the sheet. There are three components:

- The primary Save2DB tag, which defines the locations of the save-to-database control row and column, and specifies the destination table for the save. The Save2DB tag can also be used to specify optional parameters such as a custom save tag or various zero tag options.
- Column tags in the save-to-database control row. By placing the names of destination columns in the control row, you designate which columns of data to save in the sheet and where to save the data.
- Row tags in the save-to-database control column, to determine which rows of data to save or delete.

Save-to-database tag summary

Tag Type	Tag Syntax
Primary tag	`[Save2DB; TableName; CustomSaveTag; ZeroTagValue; CustomZeroTagColumn; DisableDocID; AggregateOnSave; AQName]`

Row tags	`[Save]`
	`[CustomSaveTag]`
	`[Delete]`

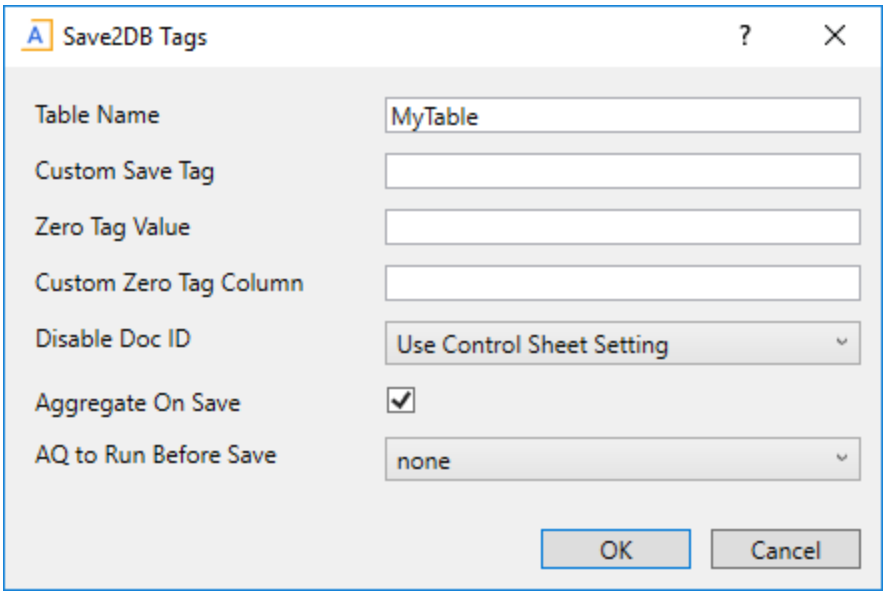
Column tags	`ColumnName`

NOTE: You can also optionally place one or more `[SaveError]` columns in the save-to-database control row to perform custom save validation. For more information, see Using custom save validation.

Defining the Save2DB tag

To define the location of the control row and column for the save-to-database process, place the Save2DB tag in any cell in the sheet, within the first 500 rows. The row containing the Save2DB tag becomes the save-to-database row, and the column containing the Save2DB tag becomes the save-to-database column.

To create the tag, you can manually type it within a cell, or you can use the Save2DB Tag helper to assist you in creating a tag. To open the helper, right-click the cell and then select **Axiom Wizards > Insert Save2DB tag**. Once the tag has been placed in a cell, you can double-click the cell to open the editor again. Keep in mind that if you edit a tag this way, the current contents of the cell will be overwritten with the revised tag—so if you have manually edited the tag to be generated from a formula, that will be lost.



Example helper dialog to create a Save2DB tag

The Save2DB tag uses the following syntax:

`[Save2DB; TableName; CustomSaveTag; ZeroTagValue; CustomZeroTagColumn; DisableDocID; AggregateOnSave; AQName]`

Parameter	Description
TableName	The name of the destination table for the save-to-database process. Data is saved from the sheet to the specified table. This is the only required parameter. The destination table cannot be a document reference table or a table using the Large Table index scheme. Document reference tables can only be updated using Save Type 3. Tables using the Large Table index scheme can only be updated using an import utility or Copy Table Data. **NOTE:** When setting up a save-to-database process in a file group file, you should never "hard-code" the table name into the tag. Instead, the table name should be looked up from the file group variables using the GetFileGroupVariable function.

Parameter	Description
CustomSaveTag	Optional. Defines a custom save tag to place in the save-to-database column, to flag the rows to be saved. Enter the tag value only—do not place the tag in brackets. For example, enter `PayrollSave` to define that as the custom save tag. The brackets are added when using the tag in the save-to-database column: `[PayrollSave]`. If you want to use the default save tag, `[Save]`, then this parameter should be omitted. The primary purpose of this optional feature is to allow you to use the same save-to-database control column with multiple save-to-database rows.
ZeroTagValue	Optional. Defines a custom zero tag to use for the purposes of zeroing associated data in the table before performing the save. If you want to use this feature, you can define the value here to be used for this save process only, or you can use the value defined on the Control Sheet for the current sheet. • If omitted, then this save process honors the **ZeroTag Value** setting, as defined on the Control Sheet. • Otherwise, specify a value to use as the custom zero tag for this save process. If a custom zero tag is defined here, the Control Sheet setting is ignored. **NOTE:** This parameter only applies if **Zero on save enabled** has been enabled on the Control Sheet.
CustomZeroTagColumn	Optional. The name of the column in the destination table to hold the zero tags. If you want to use this feature, you can define the value here to be used for this save process only, or you can use the value defined on the Control Sheet for the current sheet. • If omitted, then this save process honors the **Custom ZeroTag Column** setting, as defined on the Control Sheet. • Otherwise, specify a column name to use that column as the custom zero tag column for this save process. If a column name is defined here, the Control Sheet setting is ignored. **NOTE:** This parameter only applies if **Zero on save enabled** has been enabled on the Control Sheet.

Parameter	Description
DisableDocID	Optional. Specifies whether to disable use of the default zero tag behavior. The default zero tag behavior uses the document ID as the zero tag. If you want to use this feature, you can define the value here to be used for this save process only, or you can use the value defined on the Control Sheet for the current sheet. • If omitted, then this save process honors the **Disable DocID during Zero** setting, as defined on the Control Sheet. • If TRUE, then use of the document ID as the zero tag is disabled for this save process. The Control Sheet setting is ignored. • If FALSE, then the default zero tag behavior is used for this save process (meaning the document ID is used as the zero tag). The Control Sheet setting is ignored. **NOTE:** This parameter only applies if **Zero on save enabled** has been enabled on the Control Sheet.
AggregateOnSave	Optional. Specifies whether to aggregate duplicate values when saving data to a data table. • If TRUE or omitted (the default behavior), then rows with duplicate keys are aggregated before saving data to the database. In general this means that numeric values are summed; other values such as strings or dates must match in order to be aggregated. If non-numeric data cannot be aggregated, a save error will result. • If FALSE, then aggregation is not performed and the save process cannot contain rows with duplicate keys. If duplicate key rows are present, a save error will result. **NOTES:** • Aggregation never applies when saving data to a reference table, regardless of this setting. • If the destination table has one or more alternate key columns, then aggregation must be disabled for the save. • If the save-to-database is being used to add new records to a table with an identity key column, then aggregation must be disabled for the save.

Parameter	Description
AQName	Optional. Specifies the name of an Axiom query to run before processing the save-to-database block defined by this Save2DB tag. You can use any Axiom query that is defined on the same sheet as the Save2DB tag. When a save-to-database is executed for the file, the specified Axiom query will be run immediately before this particular save-to-database block is processed. For more information, see Running an Axiom query before a save-to-database.

For more information on using the settings that relate to the zero tag, see Using zero tags with Save Type 1.

NOTES:

- The primary Save2DB tag must be located in the first 500 rows of the sheet.
- The Save2DB tag can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For example, you may want to derive the table name for the save by using a cell reference. For more information, see Using formulas with Axiom feature tags.
- It is possible to append a number to the Save2DB tag, to specify a processing order within the sheet. See Specifying the processing order of save blocks.

Defining the database columns for the save process

For each column of data that you want to include in the save-to-database process, enter the name of the destination column in the save-to-database control row. For example: `ACCT` or `M1`. Column alias names and fully qualified Table.Column names can also be used.

Column names can be placed to either the right or the left of the Save2DB tag.

The columns listed in the control row must meet the following requirements:

- Key columns and alternate key columns must always be included in the save. These columns must always have a defined value except in one case: if you are saving new records to a table with a key identity column, the identity column can be left blank so that an identity value is automatically assigned to the new record.

- Validated columns (columns with assigned lookup columns) must be handled in one of the following ways if new records are being added to the table:
    - If a validated column has a valid default value (as set in the column properties), then you can optionally omit the validated column from the save or include it as blank, and the column will use the default value. If the column's default value is null, then the column value is set to null for that record.
    - Otherwise, the validated column must be included in the save and must have a valid defined value.

  When updating existing records, validated columns can be included or not as necessary. If the column is included but left blank, then the existing value in the column is replaced with the column's default value (assuming the default value is valid in the lookup column).
- All other columns in the destination table can be included or not as necessary. If a column is not included and the save-to-database is being used to save new records to the table, the default value for that column will be used in the new records.

  If a non-validated, non-key column is included but left blank, then the column value will be saved as "blank" if it is supported by the column (meaning empty string for string columns, nulls for columns that allow nulls). If the column does not support empty string or null, then the default value for the column will be applied.
- The columns listed in the control row must already exist in the destination table. The save-to-database process does not create columns.
- All columns listed in the control row must point to the same table, whether they are "real" column names or alias names. If you want to save to multiple tables, you must use multiple save-to-database rows.

The control row must be dedicated to containing only valid database column names. Any entries in the control row that do not match valid column or alias names for the destination table will cause an error when saving. However, known reserved tags for other Axiom features can be placed in the control row and will be ignored by the save-to-database process.

Flagging the rows of data to save or delete

For each row of data that you want to include in the save-to-database process, enter one of the following tags within the save-to-database control column. Tags must be placed below the Save2DB tag.

Tag	Description
`[Save]`	Use this tag to flag rows that you want to save to the database, assuming that you have not defined a custom save tag in the Save2DB tag. If the record does not already exist in the database (based on the key column values), then the new record will be added to the table. If the record does already exist in the database, then the existing record will be

Tag	Description
	updated (and only the columns listed in the control row will be changed).
`[CustomSaveTag]`	If you have defined a custom save tag in the Save2DB tag, then use that custom tag to flag each row that you want to save to the database. This performs the same save as the `[Save]` tag. **NOTE:** If you have defined a custom save tag for a save process, that save process *only* recognizes that unique save tag. For example, if you have defined a custom save tag of `[SavePayroll]`, only rows flagged with `[SavePayroll]` will be saved. You cannot also use the default `[Save]` tag for that save process.
`[Delete]`	Use this tag to flag rows that you want to delete from the database. Because the entire record in the database will be deleted, the target columns marked in the save-to-database control row are irrelevant in this case (except for purposes of matching the keys). There is an exception if the key column is an identity column; in that case you must also include any columns with lookup assignments. Deletions are processed *after* data inserts and updates. Therefore if you have two rows with the same key codes, and one row is configured to save while the other is configured to delete, the save would be performed first, and then the record would be subsequently deleted. **NOTE:** The `[Delete]` tag is not supported for use with multipass file processing.

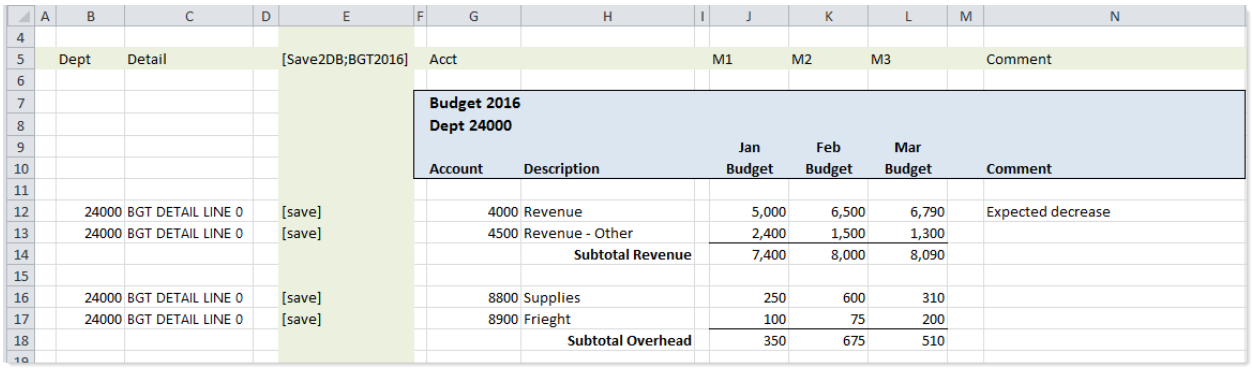
The save-to-database process only looks for rows that are marked with a valid tag; all other rows are ignored. Other entries in the control column will not cause a save error.

Save Type 1 also supports zeroing data instead of deleting records. You can use `[Save]` to save a record with zeros in the save columns, or you can enable the automatic "zero on save" process that can be used to zero "orphaned" records. For more information on this process, see How Save Type 1 works.

NOTE: The row tags can be placed in a formula if desired. For example, you might use a formula to determine whether a particular row should be saved or deleted (or not).

Save2DB tags example

The following screenshot shows example Save2DB tags in a sheet. This is not a real-life example; the intent is to show the placement of the tags.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
4														
5		Dept	Detail		[Save2DB;BGT2016]		Acct			M1	M2	M3		Comment
6														
7							**Budget 2016**							
8							**Dept 24000**							
9										**Jan**	**Feb**	**Mar**		
10							**Account**	**Description**		**Budget**	**Budget**	**Budget**		**Comment**
11														
12		24000	BGT DETAIL LINE 0		[save]		4000	Revenue		5,000	6,500	6,790		Expected decrease
13		24000	BGT DETAIL LINE 0		[save]		4500	Revenue - Other		2,400	1,500	1,300		
14								**Subtotal Revenue**		7,400	8,000	8,090		
15														
16		24000	BGT DETAIL LINE 0		[save]		8800	Supplies		250	600	310		
17		24000	BGT DETAIL LINE 0		[save]		8900	Frieght		100	75	200		
18								**Subtotal Overhead**		350	675	510		

The Save2DB tag in cell E5 defines the control row and control column for the save-to-database process (shaded in green in this example). Column names are placed in row 5, and save tags are placed in column E. When this save occurs, the rows marked with `[save]` will be saved to the database. The data to be saved is determined by the column names in row 5.

In this example, the BGT2016 table has keys of Dept, Acct, and Detail. Row 12 in this sheet will be saved to BGT2016 as follows:

- The keys that identify the record to be updated are Dept 2400 (cell B12), Acct 4000 (cell G12), and Detail BGT DETAIL LINE 0 (cell C12).
- Columns M1-M3 will be updated with the values in cells J12, K12, and L12. The column Comment will be updated with the value in cell N12.

Note the following in this example:

- Column tags can be placed to the right or the left of the Save2DB tag. This allows for greater flexibility in spreadsheet design, as some data can be placed off to the side in "work columns" versus columns that users interact with.
- Rows that are not marked with save tags are simply ignored by the save-to-database process. Notice there is no need to specifically "exclude" header rows or subtotal rows. Only rows marked with `[save]` will be included.
- If a column tag is present in the control row and a save tag is present in the control column, then the intersecting cell must contain valid data or else an error will result. For example, if cell B12 does not contain a valid department code, or if cell L12 contains string data instead of numeric data, then the save would be stopped with errors.

Using the same save-to-database control column for multiple save processes

Each instance of the Save2DB tag defines a save-to-database process for a single destination table. If you want to save data to multiple tables from within the same sheet, you must use multiple Save2DB tags and define a unique save-to-database control row for each save process.

Although each destination table must have its own unique save-to-database control row, if desired you can use the same save-to-database control column for multiple save-to-database processes.

For example, imagine that you want to save data to two different tables, GL1 and Payroll. You can place the Save2DB tags in the same column, but in different rows. Then by using the CustomSaveTag parameter, you can define unique save tags for each one—for example `[SaveGL]` and `[SavePay]`. In the save-to-database column, you would place `[SaveGL]` on rows that you want to save to the GL1 table, and `[SavePay]` on rows that you want to save to the Payroll table.

	A	B	C	D	E	F	G	H	I	J
1										
2										
3			[Save2DB;GL1;SaveGL]		Dept	Acct				
4			[Save2DB;Payroll;SavePay]		Dept	Acct	EmpID	M1	M2	M3
5										
6			[SaveGL]		200	1000		501	600	546
7			[SaveGL]		200	2000		2369	5690	2160
8										
9			[SavePay]		200	5000	6987	365	365	365
10			[SavePay]		200	6000	9764	50	50	50

Example shared save-to-database control column

Or, if you wanted to save the *same* rows of data to two different tables, you could again use a shared save-to-database control column, but instead of defining unique save tags, you could use the default `[Save]` tag to flag rows to be saved to both tables.

Using a shared save-to-database control column is only possible if all flagged rows save back to one table or the other, *or* if all flagged rows save back to both tables. If some rows save to one table, some rows save to the other table, and some rows save to both tables, then you must define two separate control columns. Once you have defined a custom save tag for a save-to-database process, that process *only* recognizes that unique save tag, and you cannot place multiple save tags in one cell.

Specifying the processing order of save blocks

Save-to-database blocks are identified in order sheet by sheet, and then by location within the sheet. By default, once a save block is identified, it is triggered for execution using a background task, so that multiple save blocks can be run concurrently as appropriate.

In some cases it may be necessary to process save blocks in a specific order. For example, you might have one block that saves a new element to a reference table, and then another block that saves data to a data table using that new element. In this case, the reference table save block must be identified and triggered for execution first, or else the data table save block will fail.

If the dependent save blocks are already in the correct order due to their placement on different sheets, or due to their placement on the same sheet, then no special considerations apply. However, if the save blocks are located on the same sheet and they are not placed in the correct order, you can manually specify the order on the Save2DB tag instead. For example:

```
[Save2DB2;BGT2021]
[Save2DB1;Project]
```

In this example, the save to the Project table will be triggered before the save to the BGT2021 table, regardless of where the tags are placed on the sheet. Because Project is a reference table, Axiom automatically waits for that save block to finish processing before triggering the save to BGT2021.

You can append any number to the end of the Save2DB tag to specify the order. If a tag exists with no number (just Save2DB), that tag is always processed first.

NOTE: The order number only affects the order in which save blocks are triggered for processing; it does not necessarily mean that the save blocks will be completed in order. The default parallel save behavior automatically handles potential dependencies for cases where the order of completion matters. For more information, see How Save Type 1 works.

How Save Type 1 works

This topic explains how save-to-database processes are triggered and executed, to help file designers configure save blocks correctly.

Order of save block processing

When a save-to-database is executed in a file, Axiom identifies the save blocks in the file and then triggers them for processing. By default, multiple save blocks are triggered and run concurrently, unless Axiom encounters a potential dependency that automatically causes processing to wait. "Save blocks" refers to Save2DB tags for Save Type 1, and SaveStructure2DB tags for Save Type 4—both save types are handled using the same process.

Save blocks are identified and triggered for execution in the following order:

- Each sheet is processed in the order they are listed on the Control Sheet.
- Within a sheet, Axiom searches for save blocks by row from left to right, starting at the first row and then moving down. If Save2DB tags are numbered, then the save blocks are processed in numeric order, otherwise they are processed in the order found.

When Axiom identifies a save block to be processed, it triggers a background task to handle the save instead of waiting for the save to finish. It then immediately moves on to identify and trigger the next save block, so that multiple saves are run concurrently. In order to automatically handle dependent save-to-database processes, the following exceptions apply:

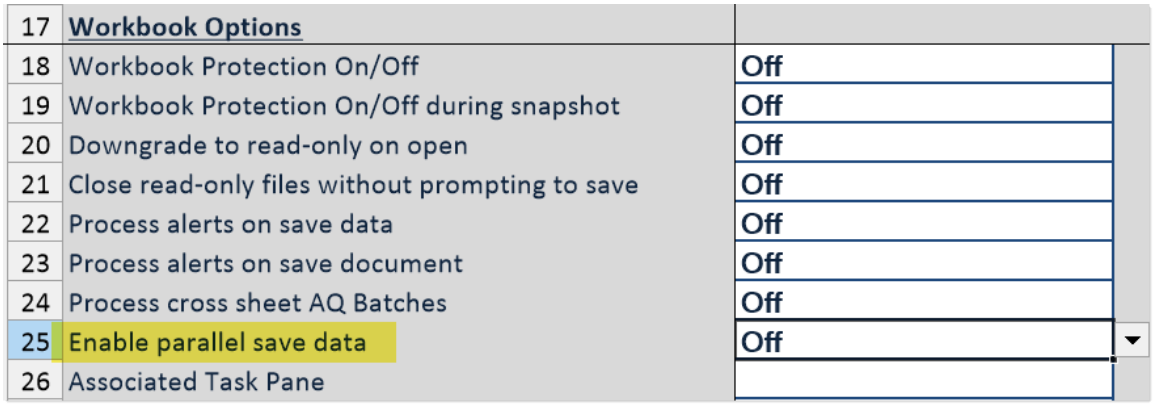
- **Reference tables**: If the destination table for the save block is a reference table, Axiom triggers the background task for the save but then waits for all current tasks to complete before moving on to the next save block. This behavior is intended to handle cases where records are saved to a reference table, and then a subsequent save references the new or updated records.

- **Save Type 4**: If the save block is SaveStructure2DB (Save Type 4) instead of Save2DB (Save Type 1), Axiom triggers the background task for the save but then waits for all current tasks to complete before moving on to the next save block. This behavior is intended to handle cases where subsequent save blocks depend on the changes made using Save Type 4.
- **Axiom query**: If the save block triggers an Axiom query to be run before the save-to-database occurs, Axiom waits for all current tasks to complete before continuing. Once all tasks are complete, Axiom runs the specified Axiom query and then triggers the background task for the save. This behavior is intended to handle the case where the Axiom query is returning records that may be impacted by a previous save block.
- **Same data table**: If the destination table for a save block is the same destination data table as a previous save block, Axiom triggers the background task for the save but does not start processing it until the task for the previous save block to the table has been completed. This behavior is intended to prevent database errors that could result if both save blocks attempted to add or update the same records at the same time.

If a particular save block encounters an error, processing stops for that block and no new save blocks are triggered for processing. However, any other save blocks that are already in process will finish processing.

If necessary, it is possible to disable the default parallel save behavior, and instead process all save blocks sequentially. In this case, save blocks are still identified in the same order, but they are processed one by one, with no overlap. If an error occurs in the current save block, all save processing stops. To disable the default parallel save behavior:

- **Per File**: To disable the parallel save behavior for a particular file, set **Enable Parallel Save Data** to **Off** on the Control Sheet.



- **System-Wide**: To disable the parallel save behavior for an entire system, set the system configuration setting **ParallelSaveEnabled** to FALSE. Generally speaking, you should only do this if instructed by Axiom Support.

NOTE: The parallel save behavior only applies when a user executes a save from within a file, to improve performance when communicating between the client and the application server. When a file is processed server-side using Scheduler and a save-to-database occurs, all saves are executed sequentially.

How Save Type 1 updates data in the database

This section describes the basic update behavior for Save Type 1. Advanced options may be used to adjust this behavior as needed.

Each Axiom file has a unique document ID. The first time a save-to-database is performed for a file, Axiom generates a zero tag based on the document ID. For each record of data saved to the destination table, this tag is placed in the SaveTagDocID column. The SaveTagDocID column must be manually created in the destination table—it is not system-generated.

When subsequent data saves are performed for that file, Axiom does the following:

1. Compares the records to be saved from the source sheet to the existing records in the destination table that are flagged with the matching zero tag in the SaveTagDocID column. If a record exists in the table but not in the sheet data, then the following columns are zeroed for that record:
    - The columns listed in the save-to-database control row (except for key columns, alternate key columns, and validated columns)
    - The SaveTagDocID column
2. Saves the sheet data to the database. For each record that was updated or inserted, the save tag is placed in the SaveTagDocID field.
3. Deletes records that are flagged to be deleted.

The zero process is intended to ensure that no old data is retained in the database from prior saves. In certain cases you may want to disable the zero process (so that no zeroing occurs on save), or modify it to use custom zero tags (so that each save process is tracked independently).

If multiple rows of data in the sheet share the same combination of key codes, then by default all non-key and non-validated fields in those rows are aggregated when saved to the database. This default behavior only applies when saving to data tables (aggregation never occurs when saving to reference tables), and can be disabled if desired by using a parameter in the Save2db tag.

Updating a record using multiple save processes

The default zero behavior described in the previous section assumes that data for each unique table record is saved from one save process in one file only. If you save data to a record from multiple files (or multiple save processes in the same file), then each subsequent save process will overwrite the zero tag from the prior process. To enable saving to the same records from multiple save processes, you must define a custom zero tag and custom zero column for each additional save process that has the same target records (advanced settings **ZeroTag Value** and **Custom ZeroTag Column**).

For example, you may have a plan file for Dept 200, which saves data to Acct 10000. And you may have a report allocation "utility" which also saves data back to Dept 200 / Acct 10000. If both of these files use the default zero behavior, then when the allocation utility is run, its zero tag will overwrite the zero tag from the plan file. However, if both save processes use a different custom zero tag and column, then one process will not overwrite the zero tag for the other process.

Custom zero tag settings can be defined per sheet (in the Control Sheet) or per save process (in the Save2DB tag). For more information, see Using zero tags with Save Type 1.

Using zero tags with Save Type 1

In most cases, Save Type 1 should be configured so that the "zero on save" behavior is enabled. If enabled, then old data from previous executions of the save process is zeroed out before saving new data, thereby eliminating "orphan" data. For more information on how the save process works, see How Save Type 1 works.

NOTE: If the Save Type 1 process is in an Axiom form (or in any file that is intended to be used as read-only with save data), then use caution before enabling the zero on save option. The data saved by an Axiom form is not persisted in the file, therefore the data saved by one user may be totally different than the data saved by another user, depending on the configuration of the Axiom form. If zero on save is enabled in this circumstance, the second user may zero the data saved by the first user.

▶ Configuring zero on save for a save process

To enable the zero on save behavior, set **Zero on save enabled** to **On** in the Save Type 1 settings on the Control Sheet. By default, this setting is disabled, so you must explicitly enable it if you want to zero data as part of the save process. Note that this will enable the zero on save behavior for all Save Type 1 processes for that sheet.

If enabled, the zero on save behavior uses a dedicated column in the destination table to flag rows of information as coming from a particular document or save process. This column must be manually created in the destination table; Axiom does not automatically create it. Within the column, a *zero tag* is used to flag the rows.

By default, the zero on save behavior looks for a column named SaveTagDocID, and uses the document ID as the zero tag. This behavior is sufficient if the data for a particular record is only saved from a single process in a single file. If a particular record may be updated from more than one file, or from multiple save processes within a single file, then you must use a custom zero tag and custom zero column (either alone or in combination with the default document ID behavior). This is necessary to prevent one save process from overwriting the zero tags of another save process, for the same record.

When you enable Save Type 1 for a file, you must create one (or both) of the following columns to hold the zero tags:

Column	Data Type	Description
SaveTagDocID	Integer	By default, the zero tag is the document ID, and the tags are saved to a column named `SaveTagDocID`. This column must be manually created in the destination table. Axiom will use this column as long as **Disable DocID during Zero** is set to **Off**, which is the default setting.
SaveTagCustom (or any custom name as specified in the zero settings)	String (any length)	If you want to use a custom zero tag and column, you must complete the following settings (either on the Control Sheet or in the Save2DB tag): • Define a value for the **ZeroTag Value**. • (Optional) Define a column name in **Custom ZeroTag Column**. If no column name is listed for the Custom ZeroTag Column setting, then by default Axiom looks for a column named `SaveTagCustom`. If a column name is specified, then Axiom uses the specified column. In either case, the column must be manually created in the destination table. • (Optional) Set **Disable DocID During Zero** to **On**. Turning this setting to On means that the default behavior of saving the document ID to the SaveTagDocID column is disabled. If desired, you can leave this setting at Off, and therefore use *both* the default column and the custom column. **NOTE:** If defining this setting in the Save2DB tag, use the Boolean value TRUE instead of On.)

Defining zero on save settings on the Control Sheet or in the Save2DB tag

If you want to use a custom zero tag and column, you can define these settings in the Control Sheet or in the Save2DB tag.

- If these settings are defined on the Control Sheet, then they apply to all Save Type 1 processes in the sheet. This may not be the desired behavior if you have more than one save process within a sheet. Remember that if you have multiple save processes in a sheet that save back to the same destination table, using the same zero tag settings will result in one save process overwriting the zero tags of the other save processes, so in this case you must define the settings in the Save2DB tag.

- If these settings are defined in the Save2DB tag, then they apply to that save process only. This is the required behavior if you have multiple save processes in the same sheet that save back to the same destination table.

  Any custom zero tag settings defined in the Save2DB tag will override the equivalent setting in the Control Sheet. For more information on how to define these settings in the tag, see Placing save-to-database tags in a sheet (Save Type 1).

Deciding which zero tag to use (DocID or Custom)

There are pros and cons to each approach, which should be considered carefully when setting up a save-to-database process.

If you use the document ID as the zero tag, then you will always know where the data came from. However, this approach also has the following risks:

- If a document is deleted and then re-created, any data saved from the prior document will now be orphaned, because the new document has a different document ID. This is why you should never delete a plan file if you plan to re-create it; instead you should always overwrite the existing document (which preserves the document ID).
- If you save to the same records from multiple files (or multiple saves in the same document), then the document ID will get overwritten on subsequent saves and therefore will not prevent orphaned data.

Using a custom zero tag addresses most of the risks of using the document ID, because you can define different zero tags and columns for each save process. And if the document ID of a file changes (due to deleting and re-creating it), it is irrelevant to the save process if you are using custom zero tags. However, the main risk of using a custom zero tag is that there is no guarantee that the custom tag is unique. It is possible to accidentally use the same zero tag and column for multiple save processes or different files, which introduces the possibility of overwriting zero tags from different processes and therefore creating orphaned data. If you choose to use a custom zero tag, take care to ensure that the zero tag is unique.

It is also possible to use both approaches for the same save process, meaning that records would be zeroed based on both the document ID and the custom zero tag. However, using both approaches also means dealing with the risks of both approaches. Generally, the only reason to use both approaches is if you have multiple save processes in the same file, saving to the same set of records.

Regardless of which approach you use, it is very important to have tie-out reports to test your save-to-database processes. Tie-out reports can help identify any issues around orphaned data.

Saving data using alternate key mapping

When using Save Type 1, you can use special syntax to look up a key value based on an alternate key value in the same table. This can be used to save data to a "child" table, where you want to populate a column in the child table with the value from the key column in the "parent" table.

The primary use case for this functionality is when inserting rows in a table that has a lookup to an identity table. Many identity tables have a column that serves as an alternate key for the table. This alternate key is typically generated using meaningful values (such as dimension codes and dates), and serves as the primary identifier of the record to end users instead of the auto-generated identity value. When inserting records into a table that has a lookup to the identity table, it may be more convenient to supply the alternate key and let Axiom look up the corresponding identity value.

Table prerequisites

The following table prerequisites are required for this special save-to-database syntax to work:

- The parent table is a reference table that contains a primary key and an alternate key. The primary key is typically an identity column and the alternate key is typically a string column, but other column types are also valid.
- The alternate key column in the parent table must have **Alternate Key** set to **True** in the column properties. This officially flags the column as an alternate key for the table and requires the column to contain unique values. Note that after a column has been designated as an alternate key, the System.IndexMaintenance job must be run in Scheduler to create the unique index on the column. Until this job is run, no data can be saved to the table.
- The child table must have a column with the correct data type to accept the values from the primary key of the parent table. Typically this column is a validated column with a lookup to the primary key of the parent table, but the lookup relationship is not required. This column can be a key column or a regular column.

Setting up the save-to-database for key mapping

The save-to-database to the child table must be set up as follows:

- Aggregation must be disabled within the Save2DB tag. Assuming the minimum parameters, this looks as follows:

  `[Save2DB;ChildTableName;;;;;False]`

  TIP: Once you have created the base Save2DB tag, you can double-click the tag to bring up the tag editor and disable **Aggregate on Save**. This will create the tag with the appropriate number of semicolon delimiters.

- In the save-to-database control row, instead of entering the name of the target column in the child table, enter the following syntax:

  `[Column=ColumnName;AlternateKey=Table.Column]`

  Where Column is the name of the target column in the child table, and AlternateKey is the name of the alternate key column in the parent table. The alternate key column must be specified using Table.Column syntax or an alias name.

  The save-to-database column that contains this special syntax is known as the mapped column. When populating the save-to-database rows, the mapped column should contain valid values as found in the alternate key column of the parent table.

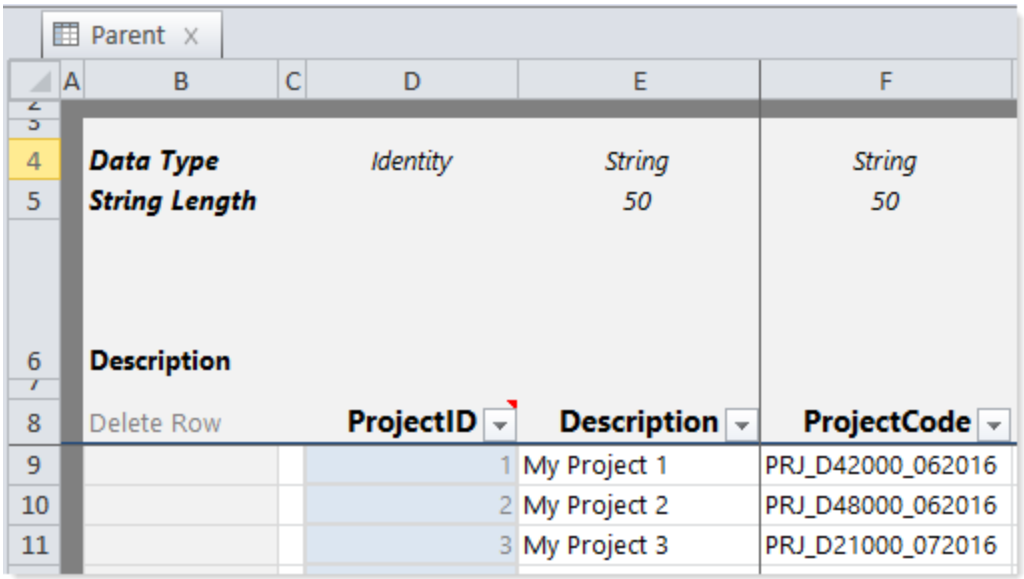
When the save occurs and data is inserted or updated in the child table, Axiom takes the value in the mapped column and finds that value in the alternate key column of the parent table. Axiom then looks up the corresponding primary key value for that alternate key value, and populates the target column of the child table with the primary key value. You can also delete records when using alternate key mapping, even if the mapped column is a key column (in which case the mapped value will be used to identify the record to be deleted).

NOTES:

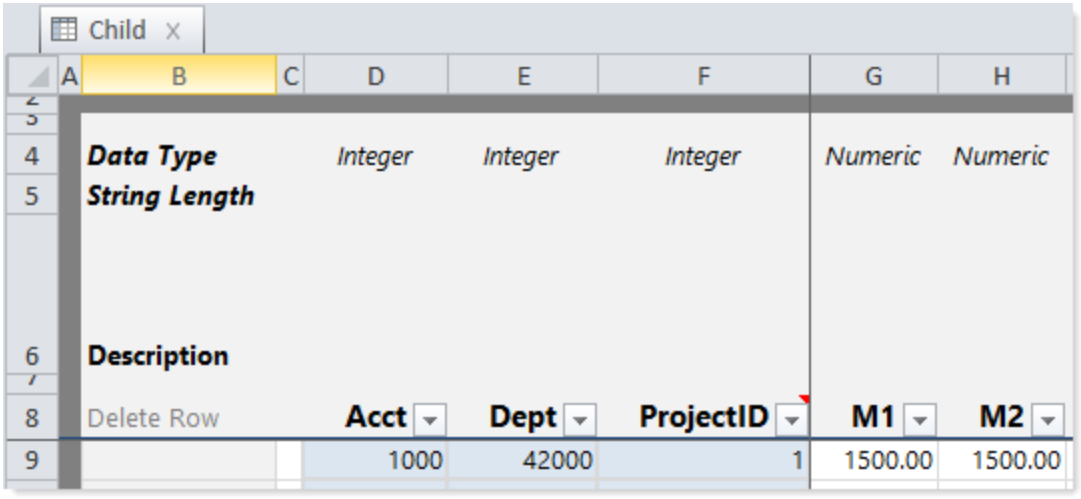
- Only one mapped column can be used in the save-to-database control row. It is not possible to map multiple columns in the save set.
- If the user performing the save has a write filter defined for the child table, all columns used in that filter must be included in the save-to-database control row.

Example

In this example, the parent table has an identity key column named ProjectID, with an alternate key column named ProjectCode (a string column).



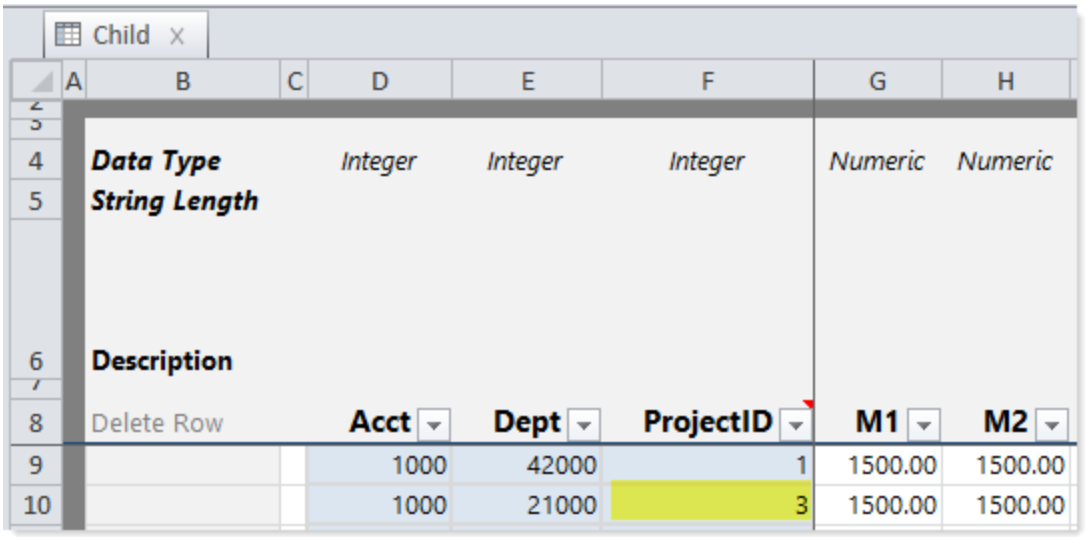
In the child table, the key column ProjectID has a lookup to Parent.ProjectID. The mapping process could also be used if ProjectID was not a key column.



In the spreadsheet, the save to database tags would be set up as follows:

	A	B	C	D	E	F	G	H
4								
5		[Save2DB;Child;;;;;False]		Acct	Dept	[Column=ProjectID; AlternateKey=Parent.ProjectCode]	M1	M2
6		[save]		1000	21000	PRJ_D21000_072016	1500	1500

When the save occurs, Axiom takes the alternate key value PRJ_D21000_072016 and finds it in the Parent.ProjectCode column. Axiom then looks up the corresponding key value in the Parent.ProjectID column, which in this case is 3. Axiom takes the value 3 and populates the Child.Project ID column with that value when creating the new record.



Running an Axiom query before a save-to-database

In some cases, you may need a particular Axiom query to run immediately before a save-to-database block is processed. The primary use case for this option is as follows:

- The file contains multiple save-to-database blocks—meaning, multiple `[Save2DB]` tags.
- The data to be saved in one of the save-to-database blocks is generated by running an Axiom query.
- The data returned by the Axiom query is affected by one of the previous save-to-database blocks in the file.

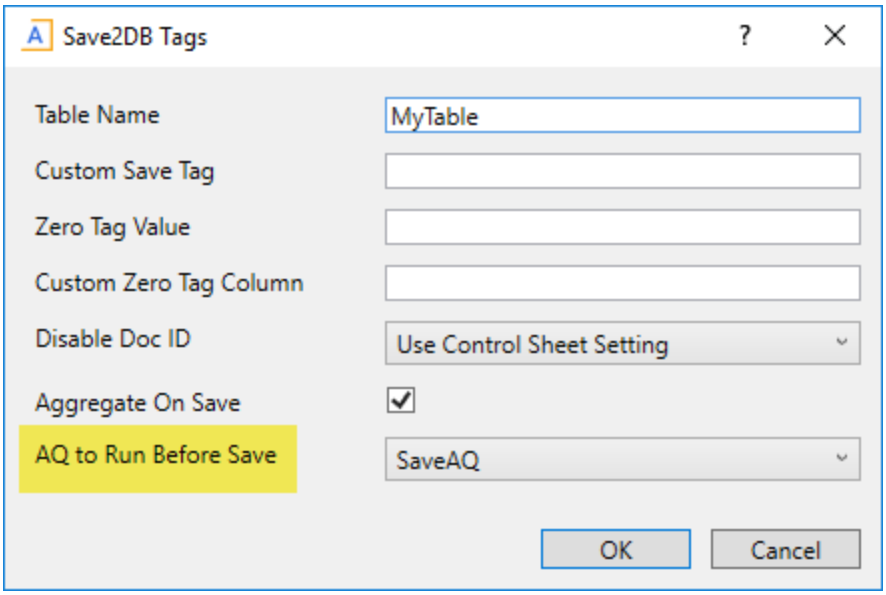
For example, imagine that you have SaveA on Sheet1, and SaveB on Sheet2. The data for SaveB is populated by an Axiom query, and that Axiom query includes data from the target table for SaveA. Under normal circumstances, SaveB would not reflect the most current data from SaveA, because the Axiom query would not be run in-between the two save-to-database blocks. Even if the Axiom query was configured to **Refresh After Save Data**, that refresh behavior means that the query is refreshed after *all* save processes are complete.

Instead, you can configure SaveB so that it triggers the Axiom query to run before SaveB is processed. This configuration is made in the Save2DB tag that controls SaveB. When the save-to-database is executed for the file, SaveA is processed first, then the Axiom query is run, then SaveB is processed.

Configuring a save-to-database to trigger an Axiom query

If you want an Axiom query to run before a particular save-to-database block is processed, enter the name of the Axiom query into the seventh parameter of the Save2DB tag. The Axiom query must be located on the same sheet as the Save2DB tag.

The easiest way to create the tag is to use the Save2DB tag helper dialog. To open the helper, right-click the cell and then select **Axiom Wizards > Insert Save2DB tag**. Once the tag has been placed in a cell, you can double-click the cell to open the editor again. Keep in mind that if the tag is constructed using a formula, then editing the tag using the helper will overwrite the formula.



Example Save2DB helper dialog with an Axiom query specified

The configuration in this example creates a Save2DB tag that looks like the following:

```
[SAVE2DB;MyTable;;;;;True;SaveAQ]
```

Immediately before this Save2DB block is processed, the designated Axiom query will be run.

Requirements and limitations

- The Axiom query must be located on the same sheet as the Save2DB tag, and the query must be active. No particular refresh behavior settings are required.
- The query can add rows with `[Save]` tags and those rows will be processed as part of the save-to-database. However, the query cannot create new Save2DB tags.
- If the Axiom query errors, then the save-to-database process fails and no further saves will be processed.

Using Save Type 3

Save Type 3 is a structured save type. It is primarily intended for use in driver files to create driver tables for file groups, but it can also be used in report files and file group utility files if appropriate. The file must be a managed file.

Save Type 3 creates a document reference table, which is a special kind of reference table that is managed within an Axiom file. Each time the data is saved to the database from the file, the table is re-created based upon the structure and data in the Axiom file. If a row or column is deleted in the document, it is deleted in the table. The table itself is never edited directly—all edits occur in the source file.

NOTES:

- Use caution when using Save Type 3 outside of driver files. Only use Save Type 3 if you want to manage the table completely within an Axiom file.
- If Save Type 3 is enabled for a sheet, no other save processes can be enabled for that sheet.

Document reference tables cannot be edited by using the **Open Table in Spreadsheet** feature. If you select to open a document reference table from the **Table** menu, Axiom opens the associated source file for the document reference table, instead of opening the spreadsheet table editor.

Enabling Save Type 3 for a sheet

To enable Save Type 3 for a sheet, complete the following settings in the **Save to Database Setup** section on the Control Sheet. Remember to add the sheet name to the Control Sheet if it is not there already.

NOTE: The save-to-database settings are protected on the Control Sheet. You must unprotect the sheet using **Advanced > Protect > Worksheet** before you can configure a save-to-database process.

Setting	Description
Save Type 3 Enabled	Enter or select **On**.
Table Name for Save Type 3	Enter the table name for Save Type 3. The name must be unique—no other sheets in this file or any other file can use the same table name. When data from this sheet is saved to the database, it creates or re-creates a table with this name.
Table Folder for Save Type 3	Specify a folder location for the table in the Table Library. For example, you might have a folder named Drivers, or a folder for each file group. If the folder does not already exist, it will be created. When entering the folder location, do not place a backward slash in front of the first folder name. For example, enter `Drivers` or `BudgetData\Drivers`, but do not enter `\Drivers`. The folder location is already assumed to be within the Table Library, so you do not need any additional folder path information. If this setting is left blank, the table is saved to the root of the Table Library.

Table names for Save Type 3 must be unique. If two different sheets use the same table name, then whichever sheet is saved to the database last will overwrite the table. Make sure that all sheets configured to use Save Type 3 have a unique table name, regardless of whether they are in the same file or different files.

To avoid this issue, it is recommended to use dynamic table names. For example, you can use the GetFileGroupVariable function to return a table name defined in the table variables for the file group.

Sheet structure for Save Type 3

Sheets using Save Type 3 must be structured in the following manner:

- Key codes must be placed in Column A. Cell A1 must contain a column title, such as "KeyCode".
- All other table columns must be defined as follows:
  - Column names must be placed in Row 1.
  - Column type must be placed in Row 2. You can specify any valid column data type except Identity. In mixed language environments, the English column type name must be used to ensure that the name will be recognized in all environments.
  - If the column type is string, the maximum string length must be placed in Row 3. The string length cannot exceed 4000. If the column type is not string, row 3 must be left blank.

NOTE: The column name, data type, and string length are the only column properties that can be set for columns in document reference tables. All other column properties will use the default setting as appropriate.

Save Type 3 evaluates the entire used area, so row and column entries do not have to be contiguous—any blank rows and columns are ignored when creating the database table.

Security access and Save Type 3

By default, when you create a table by using Save Type 3, all users have "full access" to the table through the Everyone role. This means that all users can query the data held in the table; it does *not* mean that users can access the source file. Access to the source file is controlled by the applicable file access setting in security, depending on what type of file the source file is.

In most cases, this is the desired level of access and no security changes are necessary. However, if needed you can modify the Everyone role to remove access to the table, and then grant access to the specific users and roles that need to access the table.

Table properties of document reference tables

Because document reference tables can only be managed within a spreadsheet using Save Type 3, it is not possible to directly edit the table properties using the Table Manager. Many of the table properties that are configurable for data tables and reference tables do not apply to document reference tables. For example, document reference tables cannot belong to table types, and do not have a current period. Most of the remaining table properties are set for the document reference table by default and cannot be changed.

The table name and folder for a document reference table are defined in the Save Type 3 settings within the source file. All tables created using Save Type 3 are automatically classified as document reference tables. Additionally, document reference tables are always audited, unless the in-memory table feature is enabled for your system. Auditing is automatically disabled on document reference tables if they are being stored in-memory.

Using in-memory document reference tables

Document reference tables can be created as in-memory tables. In-memory tables are stored in the main memory of the database server versus on disk (a "backup" version of the table is stored on disk for recovery purposes). Reading data from memory and writing data to memory can result in significant performance gains over disk-based table access.

Storing document reference tables in-memory may improve performance for systems with large document reference tables (around 5,000 values or greater). Systems with smaller document reference tables are less likely to realize any noticeable performance benefit.

Requirements and limitations

- In-memory tables are not supported for Axiom Cloud systems.
- The in-memory feature must be enabled for the system using the Software Manager before the document reference tables can be stored in-memory.
- In-memory tables are not audited. If the in-memory feature is enabled, auditing is disabled for document reference tables.

Enabling in-memory tables for your system

In-memory tables are enabled using the Axiom Software Manager. For more information on the Software Manager, see the *Installation Guide*.

1. In the Software Manager, click **Database Manager > Advanced Options**. The Software Manager must be run on the Axiom Application Server in order to access this option.
2. In the **Database Server Details**, verify that the **Server Name** and the **Database Name** are correct for the Axiom database that you want to configure. If not, enter the appropriate information.

   NOTE: You may be prompted to enter credentials for the specified database server, if those credentials have not already been stored in the Software Manager.

3. Select the check box for **Enable In-Memory Tables**.
4. Click **Apply**.

Enabling this option causes the following changes to be made to your database server:

- The Axiom database is enabled for Memory Optimized Data.

- A portion of the database server's memory is dedicated for use of in-memory tables. Currently this percentage is set at 3%.

After enabling the feature in the Software Manager, you must recycle the Axiom application pool so that the Axiom system recognizes the change. This is done on the Axiom Application Server, using the IIS Manager.

Saving tables in-memory

Once the in-memory feature has been enabled for your system, document reference tables will be created in-memory the next time they are saved. You should go through each one of your existing Save Type 3 source documents and perform a save to trigger the change to in-memory.

If the in-memory feature is ever disabled in the future, all document reference tables should be saved again to trigger the change back to disk storage.

Using Save Type 4

Save Type 4 provides an alternate method for performing certain system administration tasks. For example, instead of creating or editing column aliases within the table editor, you can modify them from within a spreadsheet interface using Save Type 4.

You can use Save Type 4 in any managed Axiom file, but it is most typically used in reports. Save Type 4 is configured on a per sheet basis. Each sheet can have multiple save processes.

To set up Save Type 4 for a sheet:

1. On the Control Sheet, configure the following settings:
   - If the sheet is not already set up on the Control Sheet, enter the sheet name into one of the definition columns.
   - In the **Save to Database Setup** section, set **Save Type 4 Enabled** to **On**.

   NOTE: The save-to-database settings are protected on the Control Sheet. You must unprotect the sheet using **Advanced > Protect > Worksheet** before you can configure a save-to-database process.

2. In the sheet, define the save-to-database control row and control column by placing the tag `[SaveStructure2DB;TableName;CustomSaveTag=Name]` in any cell within the first 500 rows of the sheet. The CustomSaveTag parameter is optional and can be omitted if you want to use the default save tags.

   The following table names are valid for SaveStructure2DB:

Table Name	Save Type 4 Behavior	More Information
Axiom.Aliases	Create, edit, or delete column aliases	See the *System Administration Guide*.
Axiom.CalculatedFields	Create, edit, or delete calculated fields	See the *System Administration Guide*.
Axiom.Calculations	Create, edit, or delete calculation types	This is an advanced function that should only be performed by Axiom consultants or advanced users.
Axiom.Columns	Create, edit, or delete table columns	See the *System Administration Guide*.
Axiom.ColumnSequences	Create, edit, or delete column sequences	See the *System Administration Guide*.
Axiom.ColumnSequenceItems	Create, edit, or delete column sequence items	See the *System Administration Guide*.
Axiom.FileGroupAliases	Create, edit, or delete file group aliases	See the *File Group Administration Guide*.
Axiom.FileGroupVariables	Create, edit, or delete file group variables	See the *File Group Administration Guide*.
Axiom.Notifications	Update notification status (read / not read)	See the *System Administration Guide*.
Axiom.Principals	Create or edit users in security	See the *Security Guide*.
Axiom.ProcessActions	Manage active tasks for a process	See the *Plan File Process Guide*.
Axiom.ProcessInstances	Change the process initiator for a plan file	See the *Plan File Process Guide*.
Axiom.Roles	Create or edit roles in security	See the *Security Guide*.
Axiom.SystemConfiguration	Edit system configuration settings	See the *System Administration Guide*.
Axiom.Tables	Create or edit tables	See the *System Administration Guide*.

The row containing the tag becomes the control row, and the column containing the tag becomes the control column.

3. In the save-to-database control row, for each column to be saved, enter the target column name from the specified table.
4. In the save-to-database control column, enter a row tag for each row that you want updated.

   The specific row tags depend on the table that you are saving to. All Save Type 4 processes support `[Save]` at a minimum; other tags may also be supported. See the individual topics on updating each table for more information.

   If no row tag is placed in this column, then the row will be ignored during the save-to-database process.
5. If desired, set up a `[SaveError]` column to perform custom save validation on rows to be updated. For more information, see Using custom save validation.

If you want to use a Save Type 4 process to edit or update existing values, you can use an Axiom query to bring in the current set of values from the table, then modify the values in the sheet before saving back to the database. For more information, see Using an Axiom query to return system information.

How Save Type 4 blocks are processed

When a save-to-database is executed for a file, the Save Type 1 and Save Type 4 blocks are handled as part of the same process. Axiom locates all save blocks that are either Save Type 1 and Save Type 4 and triggers them for processing in order. For more information on how these save processes are ordered and executed, see Order of save block processing.

Using custom save validation

You can set up custom save validations for your save-to-database processes, to check for issues in addition to the built-in data validation performed by Axiom. These custom validations can be used to check for data that is "invalid" due to a rule that your organization wants to enforce. For example, you can use custom save validations to:

- Require a comment to be entered if a particular condition is met for a row.
- Check for values that do not meet expected parameters, such as an expense that exceeds a specified limit.

The custom save validation consists of a condition to be checked, and an error message to be displayed if the data does not meet the condition. For example, if the custom save validation is intended to impose an expense limit, you can define a formula that tests the value in a particular cell against the defined threshold (say, $5000). If the value exceeds the limit, you can display a custom message to the user such as "Travel expense cannot exceed $5000." The user would then have to adjust their travel values in the spreadsheet before they could save the data in the file.

Validation conditions are defined in a reserved column that is labeled with the tag [SaveError]. When a save-to-database is performed, Axiom checks this column for errors. If any content is found in this column for rows that are configured to save to the database, then the save-to-database process is

stopped and the errors are displayed to the user. Therefore this column should be set up with conditional statements that return blank if the data meets your organization's rules, and return an error message if the data does not. For example:

```
=IF(F25<=5000,"","Travel expense cannot exceed $5000")
```

In this example the cell is blank if the value in F25 is less than or equal to $5000, so the save-to-database process is not stopped. If F25 is greater than $5000, then the error message is displayed in the cell and therefore stops the save-to-database process. The error message displays to the user in the Save Errors task pane.

Requirements and limitations

- Custom save validation only applies to Save Type 1 and Save Type 4.
- Custom save validation is defined per save process (meaning per `[Save2DB]` or `[SaveStructure2DB]` tag). Each save-to-database control row can have one or more `[SaveError]` columns to validate the data for that particular save process.
- Validation conditions are only valid on rows that are configured to save to the database (either as saved rows or deleted rows). Therefore, custom save validation is appropriate to test data that is row-specific, but not aggregate data.

  For example, imagine that you have five rows of expense data in a file, and each row is configured to save to the database. You can test a value on each individual row to see if that value exceeds a threshold, but you can't test the sum of all five rows (unless you are saving that summed row to the database, or creatively using formulas).

Setting up custom save validation

To set up custom save validation for a save process:

1. On the save-to-database control row for the save process that you want to validate, enter the tag `[SaveError]` in any column.

   If desired, you can append a column letter to the tag, to indicate the column that contains the data you are validating. For example: `[SaveError;F]`. This will be used to provide cell-specific error detail to the user; otherwise just the row with the error will be indicated. See the following section for more details.

2. Within the `[SaveError]` column, enter a conditional formula on each row that you want to check for save errors. The formula must return blank if the data is acceptable, and return an error message if the data is unacceptable. You can use any formula that you wish to test the data.

   If you are configuring save validation in a plan file, then in most cases these formulas will be defined within the individual calc methods, so that the validation conditions are present in the appropriate column when these calc methods are brought into the plan file.

NOTE: Custom save validation will only be checked for rows that are configured to save to the database. If a row does not contain a valid save-to-database tag such as `[Save]`, `[Delete]`, or a custom save tag, then that row will be ignored for purposes of save validation.

You can define multiple `[SaveError]` columns on the control row, and each one will be checked during the save-to-database process. For example, you might have two different conditions that you want to check for a particular row: did the user define a comment, and does the value for the row exceed $5000? Instead of having to define a complex IF statement to test both conditions and return the appropriate error message, you can place the comment validation formula in the first `[SaveError]` column, and the value threshold validation formula in the second `[SaveError]` column. If the row violates both conditions, then both error messages will display for that row.

Communicating errors to the user

In the Windows Client and the Excel Client, save errors are displayed in the Save Errors task pane. If an error is present within a `[SaveError]` column, the error will be communicated to the user as follows:

- If no column letter is indicated in the `[SaveError]` tag, the error message will be displayed with a link to the row that contains the error.

  For example, if the error message is from row 25, the user can navigate to row 25. Since the user is not directed to a particular cell, the error message should be detailed enough so that the user can easily find the problem and correct it.

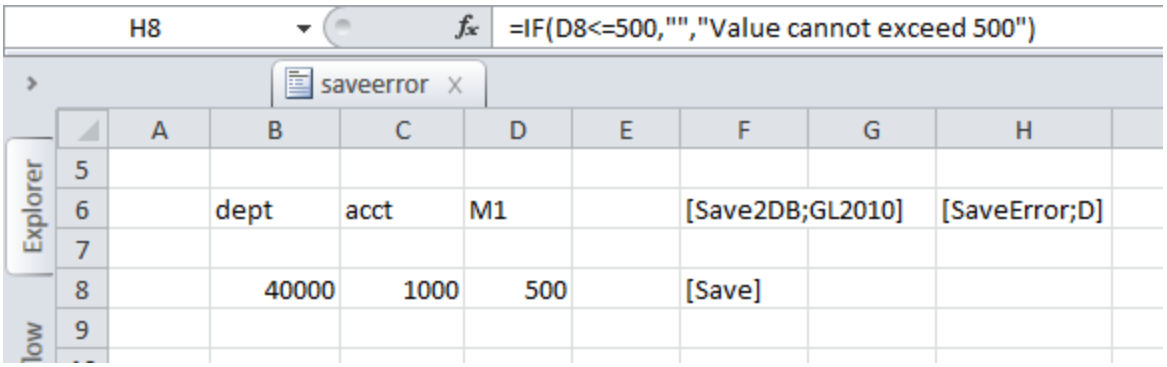
- If a column letter is indicated in the `[SaveError]` tag, the error message will be displayed with a link to the cell in that column (for the row that contains the error).

  For example, if the error message is from row 25 and the column letter in the tag is E, the user can navigate to cell E25. The intent is that if a column letter is specified, the user will be taken directly to the cell that needs attention.

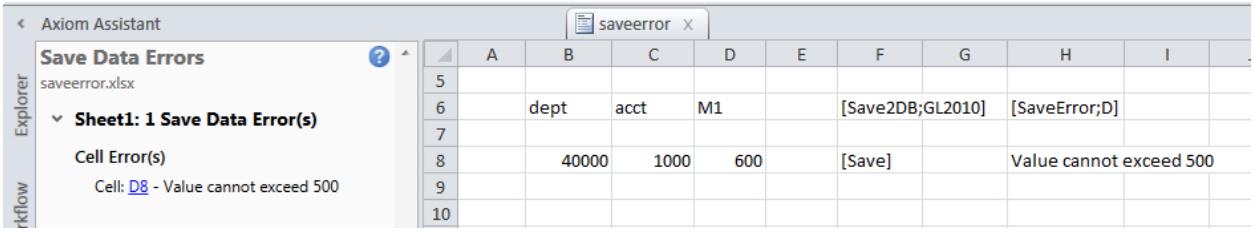
If you want to use custom save validation with Axiom forms, keep in mind that specific row and cell references are not relevant in Axiom forms, and the presentation of save errors is different. In the Axiom form environment, only the error message will be displayed to the user. The error message should be specific enough so that the user can easily find the problem area within the form without having direct navigation. You may also want to use conditional formatting on the cell so that it changes in some way when the error is triggered (for example, a red border around the problem cell).

Example of custom save validation

The following example shows a simple save-to-database process with a `[SaveError]` tag to the right of the `[Save2DB]` tag. The validation condition, shown in the formula bar, tests whether the value in cell D8 exceeds 500. Because the value is currently 500, the value meets the condition and the `[SaveError]` column is blank. When the user saves, no error will be triggered.



In the following screenshot, the value has been changed to 600, which exceeds the threshold. An error message now displays in the `[SaveError]` column, so the save-to-database process is stopped and the error message is displayed to the user in the task pane. Because the `[SaveError]` tag contains the column letter D, the error directs the user to the specific cell with the problem: D8. If no column was indicated in the tag, then the user would be directed to row 8.



9

File Output Setup

Axiom provides a variety of file output options to share data with people throughout your organization. This section explains the file setup to use these features.

- **Print view setup**: You can set up one or more custom print views for each sheet in an Axiom file. These print views can be associated with sheet views to automatically hide and/or format rows and columns in the print copy.
- **Snapshot setup**: Users can take snapshot copies of Axiom files without requiring any advance setup. However, if desired you can flag certain rows and columns in the sheet to be deleted in the snapshot copy. The primary use for this would be to delete "work areas" or Axiom query artifacts that are no longer necessary in the snapshot copy.

Axiom files can also use *file processing*, which allows you to automate certain processes for a file, including file delivery. For more information, see the *File Processing Guide*.

Setting up print views for Axiom files

You can set up one or more print views for an Axiom file. Using these print views, you can:

- Hide and/or size rows and columns for printing, by associating the print view with an existing sheet view
- Set the print orientation and paper size for the print view
- Set the repeating rows and columns for the print view
- Set the "fit to pages" and zoom for the print view
- Set headers and footers for the print views
- Set page breaks on specific rows and columns

Print views are defined on a per sheet basis. When a user wants to print an Axiom file, they can select from any defined print view in the file. If a sheet does not have any defined print views, then a "default" view is available, which uses the Excel print settings defined for the spreadsheet.

It is not required to set up print views in order to print an Axiom file—you can always use the standard spreadsheet printing features to print. The Axiom print views provide you with enhanced printing features such as the ability to print different "views" of a sheet.

For more information on how to print a file using print views, see the *Desktop Client User Guide*.

NOTES:

- The printing feature does not use the `[DeleteRow]` and `[DeleteColumn]` tags. Those tags are only used when creating snapshot copies. To hide rows and columns for a print view, use a sheet view.
- The Axiom print feature cannot be used on Control Sheets.

Print tag summary

Tag Type	Tag Syntax
Primary tag	`[Print; PrintName; ViewName; Orientation; RepeatRows; RepeatColumns;FitToPagesWide; PercentZoom; LeftHeader; CenterHeader; RightHeader; LeftFooter; CenterFooter; RightFooter; PaperSize]`
Row tags	`[Pagebreak]`

Defining a print view

You define a print view by placing the Print tag in the first 500 rows of a sheet. You can manually type the tag into the cell, or you can use the **Insert Print View** wizard to create a tag.

To define a print view using the wizard:

1. Right-click the cell where you want to place the tag, and then select **Axiom Wizards > Insert Print View**.
2. In the **Print View Options** section, complete the following:

Parameter	Description
Print View Name	Enter the name of the print view. This is the name that will display to users in the print dialog.
View Name	If desired, select the name of a sheet view to apply when printing. You can select from any sheet view that is defined within the current sheet. If you specify a sheet view, the view settings will be applied to the print copy. For example, if the view is configured to hide columns or rows, those columns and rows will be hidden in the print copy. Row and column sizing is also applied. For more information on sheet views, see Defining sheet views for a sheet.

Parameter	Description
Paper Size	Select one of the following: **Use Printing Default**, **Legal**, or **Letter**. If **Use Printing Default** is selected, then the current print settings defined for the sheet are used. **NOTE:** When editing the tag manually, this parameter should be left blank if you want to use the printing default.
Orientation	Select one of the following: **Use Printing Default**, **Landscape**, or **Portrait**. If **Use Printing Default** is selected, then the current print settings defined for the sheet are used. **NOTE:** When editing the tag manually, this parameter should be left blank if you want to use the printing default.
RepeatRows	Optional. The rows to repeat at the top of the page. Enter the rows as a range; for example: `1:3`. If left blank, and a freeze panes setting is defined on the Control Sheet, then the frozen rows are used as the rows to repeat. If freeze panes is not set, then the current print settings for the sheet are used.
RepeatColumns	Optional. The columns to repeat at the left of the page. Enter the columns as a range; for example: `A:C`. If left blank, and a freeze panes setting is defined on the Control Sheet, then the frozen columns are used as the columns to repeat. If freeze panes is not set, then the current print settings for the sheet are used.

3. In the **Scaling** section, complete the following:
   - **Fit to Pages Wide**: If you want the print copy to scale to a certain number of pages wide, select this option and then type in the number of pages (such as `1` to fit to a single page).
   - **Percent Zoom**: If you want the print copy to scale to a particular zoom percentage, select this option and then type in the percentage. Specify the number without a percent sign. For example, to zoom by 90%, specify 90.

   If you want to use the spreadsheet's native print settings to define the scaling, then do nothing. It does not matter which radio button is selected, as long as the inputs for each item are both blank.
4. In the **Headers and Footers** section, define header text as desired. You can define left, center, and right text for the header and for the footer.

   Any header or footer item left blank will use the current print settings defined for the sheet.

NOTE: Formulas and cell references cannot be entered into this dialog—for example, if you wanted to use GetFileGroupProperty or a cell reference to display the current plan code in the header or footer. However, after the tag has been placed into the cell, you can manually edit it to use a formula.

TIP: If you intend to convert the tag to a formula, you may want to enter "placeholder" text into the field that you plan to convert to a formula or cell reference. This will make it easier to find the appropriate place in the tag when you manually edit it later.

5. Click **OK** to place the Print tag into the current cell.

Once the tag has been placed into the cell, you can edit the tag manually, or you can use the wizard to make further edits. To edit an existing tag using the wizard, double-click the cell. Keep in mind that if you edit a tag this way, the current contents of the cell will be overwritten with the revised print tag—so if you have manually edited the tag to be generated from a formula, that will be lost.

If you want to generate the print tag using a formula, then you can use the wizard to create the starting point, and then edit the cell to create the formula. Any subsequent edits must be made manually. Note that the starting bracket and the identifying Print keyword must be whole within the formula—meaning, the text "[Print" must be identifiable within the formula for Axiom to recognize it as a print tag; the rest of the tag can be generated using cell references and concatenation.

Setting page breaks for the print view

There are two ways that you can set page breaks for a print view:

- You can flag rows with a `[Pagebreak]` tag within the print view itself. These tags must be placed in the control column designated by the placement of the Print tag.
- If the Print tag references a sheet view, then you can place the Pagebreak tags within the sheet view. When using this approach, you can specify page breaks on both rows and columns (horizontal and vertical page breaks). For more information, see Defining sheet views for a sheet.

Page breaks are inserted *before* the row or column that is flagged with a Pagebreak tag. This means that the next page will start with the flagged row or column.

If you have Pagebreak tags defined in both your print view and in the sheet view, then Axiom will honor both sets of page breaks.

NOTE: If "fit to pages" is used for the print job, then page break tags are ignored. If page break tags are being ignored but no scaling behavior is specified in the print view, remember that the spreadsheet's native print settings apply if no scaling is defined in the print view.

Printing design considerations

- If a sheet has a defined print area, that print area is always honored by the Axiom print option. For

example, in templates/plan files, the far right of the sheet often contains many "work columns" that you would not want to include in normal end user printing. You do not need to use a view to omit all of those columns from the print job; instead you would set the print area to the "visible" area of the plan file. The defined print area will always be printed, with the exception of any rows or columns that are hidden via the referenced view.

- If no print area is defined, then the used range defines the overall print area. However, the freeze panes and/or the repeating rows and columns settings define the start of the print area.
- All other printing features that are not controlled by the Print tag are inherited from the Excel print settings for the sheet.
- If a print view uses a sheet view, and that sheet view is not available when the user opens the Print Sheets dialog (for example, if the View tag is constructed using a conditional formula and the tag is not currently active in the sheet), then the corresponding print view will not display in the dialog.

Example print views

```
[Print;BudgetSummary;Summary]
```

This example defines a print view named "Budget Summary," which prints the sheet using the settings for the Summary sheet view.

```
[Print;BudgetSummary;;Landscape;20:25;F:H;1]
```

In this example, no sheet view is used, so the ViewName parameter is delimited with an "empty" semicolon. The print view is set to use Landscape orientation, with repeating rows and columns of 20:25 and F:H, with the print area fit to one page.

```
[Print;Payroll;NewHire;;;;;90]
```

In this example, the NewHire sheet view is used to print just the section of the Payroll sheet with new hires. All of the printing settings are taken from the spreadsheet except for the zoom, which is set to 90%.

```
="[Print;BudgetDetail;Detail;;;;;90;;"&D25&";;;;;Legal]"
```

This example is built using a formula so that the center header can read the current plan code from cell D25 (for example: "Dept 22000"). Once the tag is converted to a formula, you cannot edit the tag using the wizard (or else the formula will be overwritten with the plain tag).

This example also uses the PaperSize parameter to specify Legal-sized paper.

Configuring snapshot options for Axiom files

Axiom has several processes that create a "snapshot copy" of a file. When you take a snapshot copy of a file, the copy is converted into a regular Microsoft Excel spreadsheet that can be viewed outside of Axiom.

These snapshot options apply to the following processes that create snapshots:

- Snapshot File
- E-mail Workbook
- File Processing (Save snapshot)

Deleting rows and columns for snapshot copies

When designing an Axiom file, you can flag certain rows and columns in the spreadsheet to be deleted in the snapshot copy. This may be useful to delete things like Axiom query artifacts or work rows and columns that are no longer needed in the snapshot copy.

The `[DeleteRow]` and `[DeleteColumn]` tags can be used to flag rows and columns to be deleted in the snapshot copy. You can place these tags anywhere in the sheet. There is no control row or control column. Other Axiom processes will ignore these tags.

- Any row that contains a `[DeleteRow]` tag will be deleted in the snapshot copy.
- Any column that contains a `[DeleteColumn]` tag will be deleted in the snapshot copy.

NOTE: Any cell that contains a full tag (text with opening and closing brackets) will be processed. It is not required to match the entire cell contents. For example, a cell with text like `x[DeleteRow]` will still be processed for deletion. If you want to use a formula to toggle the deletion on or off for a column or row, then the "off" state should resolve to a blank cell (or to text such as NoDelete, but remember that text will now be present in the snapshot).

Axiom first gathers the list of rows and columns to be deleted, and then performs the deletions. Therefore, you can place DeleteColumn tags in the same row as a DeleteRow tag, and all flagged deletions will still occur.

If the sheet is configured to freeze panes via the Control Sheet, Axiom attempts to reset the frozen panes after the row and column deletions so that it is in the same relative place.

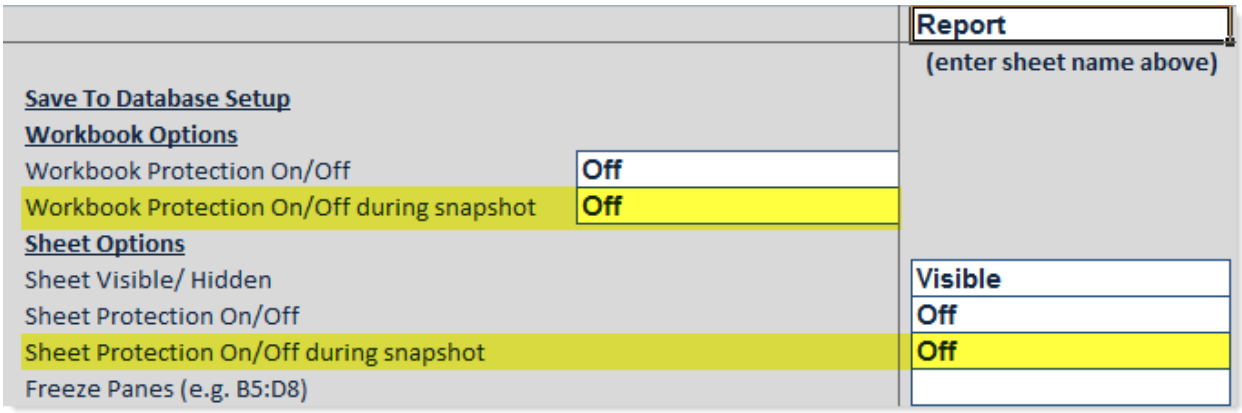
IMPORTANT: Merged cells can cause issues during the snapshot process. If possible, we recommend avoiding use of merged cells. If you must use merged cells, then any delete tags that impact the merged cells must encompass the entire merged area. For example, if you have merged cells F10 and G10, and you place a `[DeleteColumn]` tag in column F, then you must also place a `[DeleteColumn]` tag in column G. Otherwise, the snapshot process will fail because it cannot delete only a portion of the merged cells.

Applying protection to snapshot copies

You can specify whether workbook and worksheet protection are applied when a user takes a snapshot copy of a file. This protection is configured independently from the protection settings that apply when the "live" file itself is opened. You can choose to protect the live file but not the snapshot, or vice versa, or you can apply the same settings to both the live file and the snapshot.

The following settings control protection for snapshot copies:

- **Workbook Protection On/Off during snapshot**: This setting is located in the **Workbook Options** section of the Control Sheet. By default it is set to Off, which means snapshot copies will not have workbook protection applied.
- **Worksheet Protection On/Off during snapshot**: This setting is located in the **Sheet Options** section of the Control Sheet. By default it is set to Off, which means snapshot copies will not have worksheet protection applied. This setting must be defined per sheet.



NOTE: By default, these settings use a formula so that they will automatically inherit whatever is set for the "live file" setting directly above. You can override this formula if desired and choose a different protection setting for snapshot copies.

10

Data Drilling

Axiom provides several different options to drill through the data in a spreadsheet Axiom file.

- **Drill down.** You can "drill down" a data row to view the data at a different level of detail.
- **Drill through.** You can "drill through" the data in a specific cell to view the associated sub-GL detail or transactional data.
- **Custom drilling.** You can drill data using Axiom's custom drilling feature.

For more information on how to perform a drill while in an Axiom file, see the *Desktop Client User Guide*.

Drilling in Axiom forms and web reports

The information in this section applies when setting up drilling for a spreadsheet Axiom file to be used in the Desktop Client. You can also set up drilling for Axiom forms and web reports, but the setup and behavior is different. For more information, see the following:

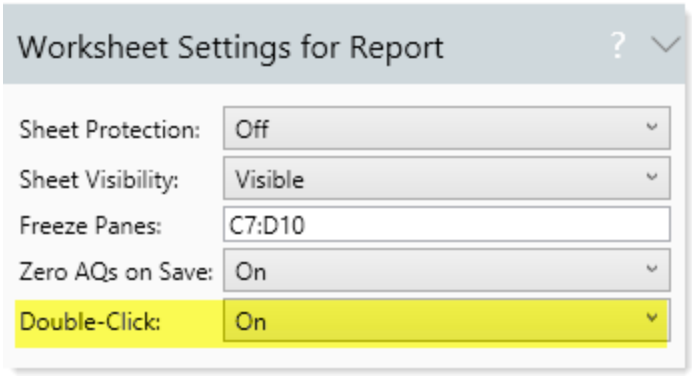
- In Axiom forms, you can set up drilling for a Formatted Grid component or a Data Grid component. For more information, see the *Axiom Forms and Dashboards Guide*.
- In web reports, you can set up drilling for a Data Grid component. For more information, see the *Web Reports Guide*.

Enabling double-click drilling

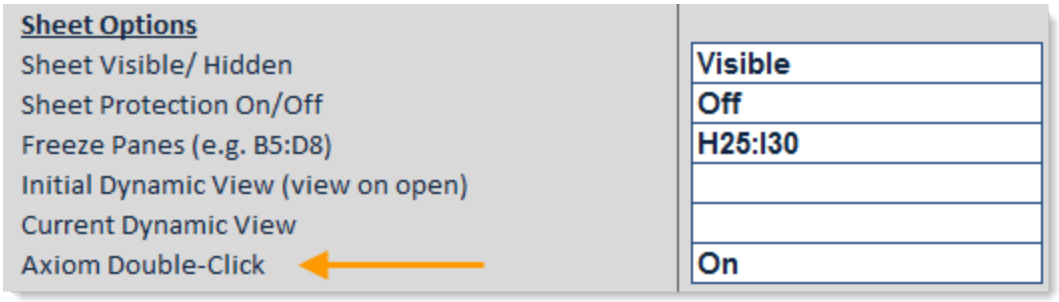
If you want users to be able to drill values in a spreadsheet Axiom file by double-clicking, you must enable double-click drilling for the file. Otherwise, users can initiate drills by using the **Drill** menu in the **File Options** group of the Axiom ribbon tab.

To enable double-click drilling for drill down or drill through, you must enable the **Axiom Double-Click** option for the file.

- On the Sheet Assistant, this setting is located in the **Worksheet Settings** section:



- In the Control Sheet, the setting is located in the **Sheet Options** section:



To enable double-click drilling for custom drilling, you must enable the option **Double-click to drill** on the Drilling Control Sheet.

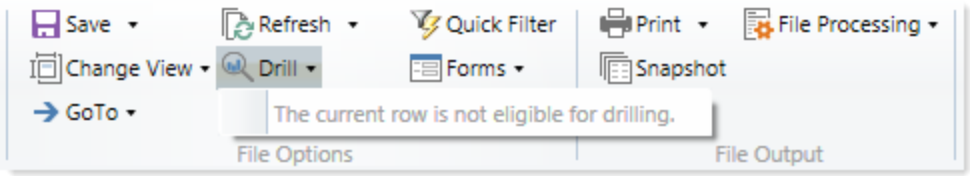
When a user double-clicks a row or cell in the file, drilling options are performed as follows:

- If custom drilling is defined for the file and **Double-click to drill** is enabled for it, the custom drill will be performed. Depending on the custom drill configuration, the drill may occur with no further input, or the user may need to select drilling options.
- Otherwise, if **Axiom Double-Click** is enabled, then a dialog opens to display the available drill down and drill through options.

NOTE: Axiom supports several different actions that may be initiated by double-clicking. When you double-click on a cell, if multiple double-click actions could apply to this context, then the action performed is evaluated based on priority. For more information, see Double-click behavior.

Disabling drilling for a sheet

By default, the **Drill** button in the **File Options** group is enabled for all sheets that are defined on the file's Control Sheet. If a user clicks the Drill button while their cursor is on a row that is not eligible for drilling, then the Drill menu displays a message to that effect.



You may have sheets in a file where you never want the Drill button to be enabled, or you may only want it to be enabled for certain audiences. In this case, you can disable drilling for the sheet by setting the **Enable Drilling** option to **Off** (either directly, or by using a formula).

Sheet Options	
Sheet Visible/ Hidden	**Visible**
Sheet Protection On/Off	**Off**
Sheet Protection On/Off during snapshot	**Off**
Freeze Panes (e.g. B5:D8)	**C7:D10**
Active Cell (selected on open)	
Axiom Double-Click	**On**
Enable Drilling	**Off**
Master Sheet	

If drilling is disabled for a sheet, then the Drill button on the ribbon is disabled when that sheet is active. Users can switch between sheets that are enabled or disabled for drilling, and the button will become enabled or disabled as appropriate. If **Axiom Double-Click** is enabled for a sheet where drilling is disabled (or if **Double-click to drill** is enabled for a custom drill), then double-click drilling will be disabled as well.

Setup for Drill Down

You can drill a row in a report or a plan file to view the data at a different level of detail. For example, if the data in a row is currently displayed at the Dept.Region level, you can drill down to view it at the department level.

"Drill down" is enabled automatically on any data that meets the drilling requirements. By default, you can drill to the "bottom" of the data (showing all details, meaning all key columns), and you can manually select columns for drilling. However, for the most intuitive and useful drill-down experience, one or more hierarchies should be set up for your data.

To set up drill-down drilling, you can:

- Define hierarchies to create predefined drilling paths for specific dimensions and groupings
- Set up the original data query and sheet formatting for optimal drill results
- Enable double-click drilling for a particular sheet

This section explains the requirements for drill-down drilling, explains how the drill data and drill sheet are created, and discusses design considerations and special behaviors. This information is primarily intended for administrators and report creators, so that they can set up the system and design reports to meet their users' drilling needs.

Drill-down requirements

This topic explains the requirements to "drill down" the data in a spreadsheet Axiom file. Plan files and reports support drill-down drilling.

NOTE: You do not need to add a Drilling Control Sheet to a file in order to use the drill down features. The Drilling Control Sheet is only for the custom drilling feature.

Prerequisites

Although hierarchies are not required to perform drill-down drilling, in most cases you will want to define one or more hierarchies for your data. In addition to drilling, hierarchies are also used in the Filter Wizard for easier filter creation.

If you do *not* define hierarchies, then by default the only two drill-down options available are:

- **All Detail**: Drill to the "bottom" of the data, meaning all key columns.
- **Choose Columns**: Select any relevant column for drilling.

If you define hierarchies, then the relevant hierarchies for the data are available on the drilling menu along with the other drilling options. Users can select hierarchy levels for quick and intuitive drilling along predefined groupings.

To drill down by hierarchy, the desired hierarchies must be defined for the relevant reference tables. Hierarchies are based on the grouping columns in a reference table.

For example, you can set up hierarchies on the DEPT reference table for Geography (Country > Region > DEPT) and Management (VP > Manager > DEPT). These hierarchies will then be available when drilling data that uses DEPT as a lookup column. (If the data comes from multiple data tables, then DEPT must be a validated column in all of the data tables in order for the DEPT hierarchies to be available.)

Hierarchies are table-specific, and are managed when creating or editing a reference table. For more information on defining column hierarchies for a table, see the *System Administration Guide*.

General drilling requirements

In all cases, the sheet must meet the following requirements in order for drilling to be available:

- The sheet must be defined on the Control Sheet.
- The **Enable Drilling** option must be **On** for the sheet. This is the default setting.

If the sheet is not defined on the Control Sheet, or if Enable Drilling is Off, then the Drill button will be disabled even if drillable content exists on the sheet.

Axiom query data rows that can be drilled

You can drill down data rows resulting from an Axiom query. The query must be enabled on the Control Sheet in order to be eligible for drilling, and the **Drillable** option for the query must be set to **On** (this is the default setting).

Note the following exceptions that *cannot* be drilled:

- Horizontal Axiom queries
- Axiom queries where the primary table is a system table (such as Axiom.Aliases)

All other Axiom queries can be drilled, however, design considerations and special behavior apply to certain Axiom query configurations. See Design considerations for drilling down Axiom queries.

GetData data rows that can be drilled

You can drill down data rows that contain one or more GetData functions. This means data rows that are unassociated with an Axiom query. GetData functions can also be used within an Axiom query, but in that case the drilling requirements and design considerations for Axiom queries apply.

Note the following exceptions that *cannot* be drilled:

- Rows where any of the GetData functions are nested within a larger formula. The GetData formula must be simply `=GetData(arguments)`, or else the row is not eligible for drilling.
- Rows where the GetData functions query different data tables, and the tables do not share the same validated key columns. Basically, if the tables referenced in a row could not be queried together within an Axiom query, then the row is not eligible for drilling.
- Rows where all of the GetData functions query reference tables.
- Rows where one of the GetData functions queries a reference table that the primary data table for the row does not link to. The "primary" data table for this purpose is the first data table referenced in the row (from left to right).

All other GetData rows can be drilled, however, design considerations and special behavior apply to certain configurations. See Design considerations for drilling down GetData functions.

Design considerations for drilling down Axiom queries

The following design considerations and special behaviors apply when drilling down the data in an Axiom query. For more information on how the drill sheet itself is created, see How the "drill-down" drill sheet is created.

Axiom query setup

If the field definition row and/or the in-sheet calc method row for the query uses formulas and cell references to locations on the same sheet but outside of these rows, then you must do one of the following to ensure that the formulas work on the resulting drill sheet:

- Enclose the "setup" area and all referenced cells within frozen panes. In this case, the entire frozen area (including rows "hidden" above the freeze panes point) is copied to the drill sheet, so the formulas will still work.
- Place the referenced cells above the field definition row and in-sheet calc method. If freeze panes is not used on the sheet, then the area from the top of the sheet down to the last row of either the field definition or the in-sheet calc method is copied to the drill sheet, so the formulas will still work. If the formulas reference cells that are below the field definition or the in-sheet calc method, then the formulas will not work.

For example, if your field definitions point to other cells to dynamically construct column names (such as CYA1), then those referenced cells must either be within the freeze panes area, or located above the field definition row / in-sheet calc method (whichever one is lowest on the sheet).

Drilling behavior for certain Axiom query configurations

Note the following drilling behavior with certain Axiom query configurations:

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. Subtotals are not applied to the data. Also, if freeze panes is not set, the column headers are created using the last column name that occurs in each field definition column.
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 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 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, then that query is not required to have a defined calc method. If no calc method is defined, then the drill results will only present the data from the query itself—any formatting or formulas on the row will not be part of the drill.
Axiom queries with GetData functions in calc method	GetData functions can be used in the calc method of Axiom queries. In this case, the drilling context is the Axiom query, not the GetData functions. For more information about drilling behavior in "mixed" configurations, see Design considerations for drilling down GetData functions.
Axiom queries with GetData functions in field definition	If a GetData function is used in the field definition of the Axiom query, it can only be drilled if the function does not use any cross-sheet references. If cross-sheet references are present, the function will return an error on the drill sheet. This applies if the GetData function is in the field definition directly or if the field definition uses a formula that references the GetData function in another cell.

Configuration	Drilling behavior
Multiple data tables	If the query uses multiple data tables: • The hierarchies shown on the Drill Down menu are based on the primary table. Only hierarchies for shared lookup tables are valid for drilling in this circumstance. • The **All Detail** drill option will only be available if the tables share the exact same key columns and all of the key columns are lookup columns. • The **Choose Columns** dialog will only show columns from lookup tables.
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. Additionally, if the AxAggregate column also has a column filter, the column is not recognized as valid for purposes of drilling and does not get copied to the drill sheet.
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.

Design considerations for drilling down GetData functions

The following design considerations and special behaviors apply when drilling down the data in a GetData row. For more information on how the drill sheet itself is created, see How the "drill-down" drill sheet is created.

▶ GetData design considerations

The following design considerations apply when setting up GetData reports for drilling:

Configuration	Design Consideration
GetDatas within formulas	Whenever possible, you should construct the report so that all GetData formulas within a row are not part of a larger formula, so that the row will be eligible for drilling. If you need to perform math or other manipulations on the result of a GetData function, you can work around it by placing the GetData function by itself in one cell on the row, and then referencing that cell within the full formula. This way the data will still be drillable. However, if the GetData function is nested within a larger function, the row will become ineligible for drilling.
Freeze panes and drillable row contents	If freeze panes is set for the sheet (on the Control Sheet), then the freeze panes settings will be copied and adapted to the drill sheet. Any content "hidden" to the left of the freeze panes will be brought to the drill sheet, but will remain hidden. If freeze panes are not set on the sheet, then the entire contents of the row are displayed on the drill sheet. If you have content to the left of the data that you do not want on the drill sheet, you should configure freeze panes settings for the sheet.
Multiple data tables	If the functions in a row query multiple data tables, then the **All Detail** drill option will only be available if the tables share the exact same key columns and all of the key columns are lookup columns. Additionally, the **Choose Columns** dialog will only show columns from the shared lookup tables.
Excel Precision	If Microsoft Excel's **Set precision as displayed** feature is enabled for the workbook, then the GetData functions cannot be drilled if the cell precision is set to a different level than the stored value. Axiom will interpret the value difference as if the GetData function were within a formula, which is not a supported drilling configuration.

▶ Drilling behavior for certain GetData row configurations

Note the following drilling behavior with certain GetData row configurations:

Configuration	Drilling Behavior
GetDatas with different filter criteria statements	If the GetData functions in the row have different filter criteria statements, the columns used in each filter must be compatible with all tables referenced by the GetData functions on the row. If incompatible columns are present, an error results when attempting to drill the row. If the drill is successful, no "drill path" will be presented on the drill sheet, because a different drill path applies to each column. **NOTE:** All of the GetData functions might use the same filter criteria statement within the formula, but still have different effective filters due to use of sheet filters. This occurs when: • The GetData functions query different tables, and the sheet filter applies to some tables but not others. • Some of the GetData functions on the row use the "ignore sheet filter" parameter.
Alternate aggregations	Use of alternate aggregations in GetData functions does not prevent drilling down a row, however, any GetData functions with alternate aggregations are omitted from the drill results.
Data conversion	If all of the GetData functions in the row use the same conversion target, then that conversion target is applied to the drill down data. If the GetData functions use different conversion targets, or if some of the GetData functions have no conversion target, then no conversion is applied to the drill down data. Axiom informs you of this condition when you initiate the drill, and asks if you want to continue.

▶ Drilling behavior for "mixed" report configurations

A "mixed" report configuration is one where GetData functions are used within an Axiom query. In this case, the drilling context is the "parent" Axiom query, not the GetData functions.

Depending on the setup of the query and the GetData functions, drill results may not be as desired for mixed configurations. Specifically, report configurations that use an Axiom query as a means to build out blocks of GetData functions may not provide the desired drill results, and in some cases may be essentially undrillable unless you configure the Axiom query so that it is no longer eligible for drilling.

For example, you might have an Axiom query that is set up to query the DEPT table. The in-sheet calc method is a multi-row block of GetData functions and other formulas. When the query is run, it creates a block for each department in the table. The resulting GetData functions cannot be drilled, because the drill context is the parent Axiom query, not the GetData functions. The parent Axiom query is already at the lowest level of detail for its primary table, and therefore cannot be drilled.

If the parent Axiom query were at a higher level of detail—say, DEPT.Region—then the Axiom query could be drilled down to DEPT. If you drilled the data in this case, the drill sheet would contain the block of GetData functions for each department in the region that was drilled. However, this may not be what you wanted when you drilled. The rows in each block could be summed at an ACCT.Category level, and perhaps what you wanted was to drill a specific GetData row to the account-level detail. This level of GetData drilling is only possible if you do one of the following:

- Disable the Axiom query (**Activate** set to **Off**). In this case you would not be able to refresh the Axiom query unless you re-enabled it. However, if the only purpose of the query was to build out the GetData blocks, it might be acceptable to disable the query.
- Disable drilling for the Axiom query (**Drilling** set to **Off**). In this case you could still refresh the Axiom query, but you could not drill it.

In both cases, now that the Axiom query is no longer eligible for drilling, Axiom will check to see if another drilling context is available. You could now drill the GetData functions directly, as long as they are otherwise eligible for drilling, according to the GetData drilling requirements.

NOTE: If the query is a horizontal query, then you can always drill the GetData functions within that query. Horizontal queries are always ineligible for drill down, so by default the drilling context is the GetData functions.

How the "drill-down" drill sheet is created

When you initiate a drill down in a spreadsheet Axiom file, Axiom creates a new, temporary spreadsheet that contains a drill sheet with the drilling results.

Drill data

On the drill sheet, the drill data is returned by creating an Axiom query based on the settings in the original sheet.

If you are drilling an Axiom query, Axiom creates the drill query as follows:

- The field definition row and the in-sheet calc method are copied from the original sheet to the drill sheet. This includes any column filters or other special syntax applied to the field definitions.

  NOTE: If the Axiom query uses a calc method library instead of an in-sheet calc method, then the calc method applied to the row being drilled is copied to the drill sheet. For more information, see Design considerations for drilling down Axiom queries.

- Axiom identifies the "sum by" level of the original Axiom query and then finds the associated entries in the field definition row. If they are located at the start of the field definition row, the corresponding entries are removed.

- New entries are added to the start of the field definition row for the selected drill level, and for a Description column (if the drill level is a key column and a description column is available). There is no way to control the formatting for these drill-level columns. The selected drill level is also added to the "sum by" for the drill query.
- A data filter is created for the drill query that limits the data returned to the row that was drilled. The data filter is determined based on the keys in the data control column of the original query. If the original query had a data filter, that filter is also appended to the data filter for the drill query.

If you are drilling a GetData row, Axiom creates the drill query as follows:

- A field definition row is created that contains entries for the selected drill level, and for the columns used in each GetData function on the row.
- An in-sheet calc method is created based on the formats and formulas used in the row. Note that non-number cell formatting applied to the first cell in the drillable row will also be applied to the cells that display the drill level.
- The selected drill level is applied as the "sum by" level for the query.
- The query data is filtered in one of the following ways:
    - If all of the GetData functions in the row use the same filter criteria statement, this is applied as a data filter for the entire query.
    - If the GetData functions use different filter criteria statements, these are applied as column filters to each field definition. Additionally, all of the filters used in the row will be concatenated using OR and applied as a data filter, in order to limit the overall data query to only the applicable data.

In both cases, if a sheet filter was defined on the original sheet, that filter is also applied to the drill sheet. This also applies to any Quick Filter currently applied to the original file. Note that the "behind the scenes" Quick Filter on the original file is converted to an explicit sheet filter on the drill sheet, so the GetCurrentValue function will return "None" for the drill sheet.

Headers

Headers are created on the drill sheet as follows:

- If the original sheet has freeze panes settings defined on the Control Sheet, the entire frozen area (including rows "hidden" above the freeze panes point) is copied to the drill sheet and becomes the header area. It is assumed that the frozen section contains titles and column headers for the report that would also apply to the drilled data.
- If the original sheet does not have freeze panes settings, then Axiom inserts a basic header row that contains the column names for the data being returned. Note that if you have columns that are calculations instead of database data (like a variance column), no column title will be placed on these columns.

- In all cases, Axiom places a column title over the drill target column, to identify the items resulting from the drill. If you drilled to a key column (for example, DEPT), and the associated reference table has a designated Description column, then the Description column is also automatically displayed and titled.
- Axiom inserts a row in the header to display the drill path. If headers were copied over from the original sheet, this row displays underneath them. Otherwise, this row displays above the automatically-generated header row.

Frozen panes

Frozen panes are set on the drill sheet as follows:

- If the original sheet has freeze panes settings defined on the Control Sheet, those freeze panes are applied to the drill sheet, after adjusting for inserted rows and columns.
- If the original sheet does not have freeze panes settings, then Axiom automatically applies freeze panes to the left and top of the first data column.

NOTE: Axiom takes into account the available screen area and will only freeze columns up to approximately half of the screen width. If the inherited or automatic freeze panes settings would exceed this width, then columns will not be frozen. Columns to the left of the drill column will be hidden instead.

Subtotals

Subtotals are placed in the drill sheet as follows, depending on whether you are drilling an Axiom query or a GetData row.

If you are drilling an Axiom query, subtotals are created as follows:

- If both the in-sheet calc method and the field definition for the query are single rows, then Axiom adds a subtotal row below the drill data. If either of these settings use multiple rows, no subtotal row is created in the drill sheet.
- The subtotal row starts as a copy of the in-sheet calc method, so that any existing formulas in the row are retained (such as a variance calculation). Then, for each non-key numeric column in the drill data, Axiom adds a SUM formula to the subtotal row.

If you are drilling a GetData row, subtotals are created as follows:

- Any column that queries a numeric column from a data table has a subtotal.
- If a column uses a non-GetData formula, that formula is copied to the subtotal row.

"Numeric" in this context means any column with a data type of Numeric or Integer (all types).

Views

If a sheet view was applied to the original sheet, the same view will be applied to the drill sheet if the View tag is defined within the header area and the header area is enclosed in frozen panes. If you need to control the column widths in the drill sheet, using a view is the best option.

If the View tag is not defined within the header area or if freeze panes is not set, then the view will not be applied to the drill sheet. This may result in additional content on the drill sheet that is not visible on the original sheet, if the view is being used to hide rows or columns.

Setup for Drill Through

You can drill a value in a report or a plan file to see the sub-GL detail or transactional data associated with that value. For example, you can drill through a sales value in a report to see the detailed sales transactions associated with that value.

"Drill through" requires the following setup tasks before it is available for use in Axiom files:

- The desired sub-GL data must be imported into Axiom so that it is available for drilling.
- Drill-through definitions must be defined in Axiom to identify the mapping relationships between the source data (the data that you want to be able to drill) and the target data.

Additionally, you may want to enable double-click drilling on certain files.

This section explains the requirements for drill-through drilling, explains how the drill data and drill sheet are created, and discusses design considerations and special behaviors. This information is primarily intended for administrators and report creators, so that they can set up the system and design reports to meet their users' drilling needs.

Drill-through requirements

This topic explains the requirements to "drill through" the data in an Axiom file. Plan files and reports support drill-through drilling.

NOTE: You do not need to add a Drilling Control Sheet to a file in order to use the drill through features. The Drilling Control Sheet is only for the custom drilling feature.

Prerequisites

In order to drill through data, drill-through definitions must be defined for the column sequence / target table combinations that you want to drill.

NOTE: The target data (the sub-GL / transactional data) must have been imported into Axiom in order to be available for drilling. You cannot point a drill-through definition to data that is outside of the Axiom system.

The *drill-through definitions* map data columns to associated data in a target detail table. The mappings are made by column sequence. For example, you may have a sequence M in the GL2020 data table which holds current year actuals data. You can map each column in the M sequence to the appropriate data in the GLDetail table.

Once you have set up the definitions, you can drill on values in an Axiom file that come from one of the columns in the mapped sequence. To continue the previous example, if a report contains data for CYA1 (column M1 of the M sequence in the GL2020 table), you can place your cursor in a cell that contains CYA1 data and drill to the detailed data in the GLDetail table.

Drill-through definitions are managed in the Table Administration area (**Administration > Tables > Table Administration > Drill-Through Manager**). For more information on defining drill-through definitions for a table, see the *System Administration Guide*.

General drilling requirements

In all cases, the sheet must meet the following requirements in order for drilling to be available:

- The sheet must be defined on the Control Sheet.
- The **Enable Drilling** option must be **On** for the sheet. This is the default setting.

If the sheet is not defined on the Control Sheet, or if Enable Drilling is Off, then the Drill button will be disabled even if drillable content exists on the sheet.

Data that can be drilled

The following data values are eligible for drill-through drilling:

- Cells that contain a value from a column in a sequence that belongs to a drill-through definition. The value can come from an Axiom query or a GetData function. The query can use the literal column name or an alias.

  NOTE: Calculated fields cannot be drilled, even if the calculated field uses a sequence that belongs to a drill-through definition. The value must come directly from the literal column.

- If the value came from an Axiom query, then the query must be enabled on the Control Sheet in order to drill it, and **Drillable** must be set to **On**.
- Any filter applied to the value—such as the Data Filter for an Axiom query, or a Sheet Filter—must also be applicable to the target table.

Data that cannot be drilled

The following data configurations are not eligible for drill-through drilling:

- Horizontal Axiom queries.
- Axiom queries where the primary table is a system table (such as Axiom.Aliases).
- GetData functions that are nested within a larger formula. The formula must be simply `=GetData (arguments)`, or else the value is not eligible for drilling.

Design considerations for drill-through drilling

The following design considerations and special behaviors apply when drilling through data in an Axiom file. For more information on how the drill sheet itself is created, see How the "drill-through" drill sheet is created.

Drilling behavior with alternate aggregations

Use of alternate aggregations on an Axiom query or a GetData function have no impact on whether a cell is eligible for drill through, and no impact on the results of that drill. If the source column is eligible for drill through, the drill will be performed regardless of the aggregation setting.

Drilling behavior with data conversions

If data conversion is enabled for the Axiom query or the GetData function, the conversion does not apply to the drill through data. Data will be presented as it was entered in the target detail table.

Drilling behavior for certain Axiom query configurations

Note the following drilling behavior with certain Axiom query configurations:

Configuration	Drilling Behavior
Nested Axiom queries	You can drill through a value from a nested Axiom query. 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 context will be derived based on the Axiom query where your cursor is currently located. For example, if AQ1 builds out multiple data ranges for AQ2, and your cursor is in an AQ2 data range, then the drill context will be determined based on AQ2. In most cases, this setup will result in values that are eligible for drill-through drilling as you would expect.

Configuration	Drilling Behavior
Parallel Axiom queries	You can drill through a value from a parallel Axiom query. 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, you can only drill through values from the query that is listed first on the Control Sheet. For example, if AQ1 and AQ2 update the same set of rows, then only those values that come from AQ1 can be drilled. If your cursor is on a value from AQ2, Axiom tries to derive the data column using the AQ1 settings. Since no data column is present in the AQ1 settings, the value is not eligible for drilling. **NOTE:** If the first query context is invalid for drilling (for example, if AQ1 is disabled, or if it has no content in the field definition row), then Axiom will attempt to drill on the next relevant query—in this example, AQ2.
Multiple-row field definition	Multiple-row field definitions are not supported for use with drill-through drilling.

Drilling behavior for "mixed" report configurations

A "mixed" report configuration is one where GetData functions are used within an Axiom query. In this case, the GetData functions are eligible for drill-through as if they were stand-alone GetData functions. All normal drill-down requirements for GetData functions apply.

Axiom evaluates the Axiom query for drill-down eligibility first. Because no database column is present in the field definition row for this context, the Axiom query is determined to be ineligible for drill down. Since the parent Axiom query is ineligible, the GetData function becomes the primary drill down context and can be drilled if otherwise eligible.

How the "drill-through" drill sheet is created

When you initiate a drill through, Axiom creates a new, temporary file that contains a drill sheet with the drilling results.

Drill data

On the drill sheet, the drill data is returned by creating an Axiom query. Axiom creates the query as follows:

- The primary table for the query is the target table from the drill-through definition.
- A field definition row is created that contains the drill target columns from the drill-through definition.

- A data filter is applied to the query, based on the following:
    - The column filter in the drill-through definition that corresponds to the column being drilled in the source file. The column being drilled is determined from either the field definition row (if drilling an Axiom query) or the column parameter of the GetData function.

      For example, if you are drilling a value from the CYA1 column, and the drill-through definition has a column 1 filter of `YearMo=201101`, that filter is applied to the drill query.
    - The dimensionality of the row being drilled in the source file. The row dimensionality is determined from either the data control column (if drilling an Axiom query) or the filter parameter of the GetData function.

      For example, if you are drilling a value that displays revenue account data, the filter `Acct.Category='Revenue'` is applied to the drill query.
- Any filters applied to the original value are also applied to the drill query. This includes:
    - Any sheet filters on the original sheet.
    - If drilling an Axiom query, the data filter for the original Axiom query.
    - If drilling an Axiom query, any column filter defined on the field definition for the value being drilled.

Headers

Headers are created on the drill sheet as follows:

- The top of the sheet contains a header that identifies the sheet as a drill-through sheet, and identifies the following:
    - **Drill to**: The target table and the column filter applied to the data, both derived from the drill-through definition.
    - **Source**: The source dimensionality of the value from the original report.
- Each column of the drill data is labeled with the column name.

Freeze panes

Freeze panes are set on the drill sheet to the left and top of the first non-key column.

The key columns included and their locations depend on the drill target column order set in the drill-through definition. Typically, key columns are displayed first, in the left-hand side of the sheet, followed by the data columns. Panes are frozen after the last key column displayed in the left-hand side of the sheet.

If no key columns are displayed to the left (meaning, the first column displayed is a non-key data column), then no frozen panes are set.

Subtotals

Subtotals are placed in the drill sheet, below the drill data, but only for decimal and numeric columns. Integer columns (all types) are not subtotaled, as they may contain descriptive data (such as a transaction ID number) instead of amounts. If you want data to be subtotaled automatically in the drill sheet, it must be a Numeric column.

Custom drilling

In addition to the "built-in" drilling features for spreadsheet Axiom files (drill down and drill through), you can use custom drilling. Custom drilling is an extremely flexible feature that can be configured to meet virtually any data drilling need. It is intended for unique or complex drilling requirements that are not met by the standard drilling features.

You can set up a custom drill for any Axiom file, but the primary use is in reports.

Custom drilling in Axiom works by configuring a file to pass dimensionality and filters to a "drill target" template. The dimensionality and filters are based on the user's current context in the file, and on user selection if appropriate.

The "drill target" template is an Axiom report that has been configured and formatted to use the dimensionality and filters passed by the source report, and display the resulting data. The display can replicate the structure of the source file—for example, the same data columns, but at a different level of detail—or it can be completely different.

Setting up custom drilling requires advanced knowledge of Axiom report writing. Ultimately, you can set up the source report to pass any combination of data filters and dimensionality, and set up the drill target to display any set of data, in any format—even data from an entirely different table.

The basic setup steps are as follows:

1. Determine the level of drill that you want to achieve, and the desired output. Visualizing the drill options and what you expect to see as a result of drilling will help the setup process.
2. In the file where you want users to be able to drill, add a Drilling Control Sheet to the file, and complete the settings as desired.
3. Create the "drill target" report to display the drill data. This is a report that is configured and formatted to display the drill results.

Creating the drill target report for custom drilling

When a custom drill is executed, Axiom takes the parameter values from the Drilling Control Sheet of the source file, and passes them to the designated locations on the target output file. If a parameter has more than one possible value, the user is prompted to select the value to use.

The drill target report must be configured in a way to intelligently use these passed parameters to query data from the database. For example, if a filter criteria statement is passed, you can reference the target cell as the data filter for an Axiom query. If dimensionality is passed (for example, Dept.Region), you can reference the target cell as the sum level for an Axiom query.

The format of the drill target report is completely flexible. The target report can replicate the data columns of the source file, but be configured to use the passed filter and dimensionality to present data at a different level. Or you can add or remove data columns, or use a completely different set of data columns. The target report is essentially a brand new report, with no restrictions.

Because custom drilling is such a flexible feature, an understanding of advanced report writing concepts is necessary to set up the drilling parameters on the source sheet and to set up the target report to properly use the passed parameters.

NOTE: The Axiom queries in the drill target file do not need to be configured to refresh on open. When the output file is opened via custom drilling, the file is automatically refreshed after the drilling parameters are placed in the file.

Controlling access to the drill target report

Users must have at least read-only access to the drill target report, or else the drill will not execute because the file cannot be opened.

You may want to place your drill target files in reserved locations in the Reports Library, so that you can easily grant access to users who need it. When granting access, you can clear the **Show in Explorer** option, which means that the drill target report will be hidden when the user browses the Explorer task pane or similar "Explorer views." However, as long as the user has Read Only access, they will still be able to open the report when executing the custom drill.

If a user has read/write access to the drill target report, then the file will open with that level of access when drilling. You can suppress this behavior by enabling **Downgrade to read-only on open** on the target report's Control Sheet.

Lastly, if you don't want read-only users to be prompted to save the file when they close it, you can enable **Close read-only files without prompting to save** on the target report's Control Sheet. This provides a more streamline drilling experience as the user can drill and close repeatedly without needing to dismiss extraneous save prompts.

Using the same file as the drill source and drill target

You can set up custom drilling so that the drill target report and the source file are the same report. In this case, the **Report folder** and the **Report file to Open** are set to the same folder path and file name as the source file. The **Sheet to activate** is the sheet in the source file where the drill results are presented.

The **Reuse drill target** setting determines the behavior when a user initiates the drill:

- If **Off** (the default behavior), then a read-only copy of the source file is opened (with a number appended to the file name), and the drill results are presented in this copy. If the user drills the source file again without closing the drill copy, then another copy is opened to show the results of that drill.
- If **On**, then the source file is closed and then reopened to display the drill results. Each subsequent drill closes and reopens the source.

  NOTE: In this configuration, if the source file contains unsaved changes the user will not be prompted to save them before the file is closed and then reopened.

Drilling Control Sheet settings

The Drilling Control Sheet defines settings to drill a particular Axiom file using custom drilling. In the Drilling Control Sheet, you can define one or more drill options for each sheet in the file. For each sheet, you can specify one drill option as the "double-click" option, which means that drill will be executed automatically when a data row in the sheet is double-clicked. Otherwise, users execute drills from the **Drill** menu in the **File Options** group.

To add a Drilling Control Sheet to an Axiom file:

- On the **Axiom Designer** tab, in the **Developer** group, click **Tools > Add a Control Sheet > Drilling**. Only administrators have access to this command.

The Drilling Control Sheet is named **Control_Drilling**. This sheet is only visible to administrators and to users with the **Allow Sheet Assistant** permission. Otherwise, it is hidden by default.

The Drilling Control Sheet must be present and configured in the file that you want to drill. The target "drill output" report does not need a Drilling Control Sheet.

The top section of the Drilling Control Sheet contains general options that apply to all drills defined for the sheet, followed by several sections of defined drill options. The default Drilling Control Sheet contains settings for three drilling options per sheet.

The Drilling Control Sheet contains one column of settings, followed by several definition columns. To set up a sheet in the file for drilling, enter the name of the sheet into row 1 of the first open definition column. The sheet listed here must also be enabled for drilling on the main Control Sheet (**Enable Drilling** must be set to **On**).

▶ Sample Drilling Formula

Row 4 of the Control Sheet contains a sample formula that can be used to obtain a value based on the active row in the report sheet. You can then reference this value in the Drill Option settings, to pass it to the target file.

The sample formula uses GetRowNumber("Active") to find the row number of the currently active row, and then returns the contents of column A on that row. You would need to set up your report sheet to place row-specific values into a particular column (such as a row-specific filter), and then modify the sample formula to point to that column.

It is not required to use this formula to set up your custom drill; the sample formula is provided as a convenience.

General Options

The general options at the top of the sheet relate to enabling "double-click" drilling for the sheet.

Field	Description
Double-click to drill	Specifies whether users can drill by double-clicking cells in a report (**On**/**Off**). **NOTE:** It is not necessary to also enable **Axiom Double-Click** on the main Control Sheet.
Double-click default drill ID	Optional. If **Double-click to drill** is enabled, this setting specifies the drill option to execute when a user double-clicks a cell in a report. Enter the ID number of the option, such as `2` for Drill Option #2. If left blank, then the following occurs: • If only one drill option is active, that drill option will be executed. • If multiple drill options are active, then a drill dialog opens where the user can select which drill option to execute (as well as complete any selections for that option).
Reuse drill target	Specifies whether the drill target file is reused each time a drill is executed (**On**/**Off**). By default, this is set to Off. If you drill a report, leave the target file open, and then drill the report again, the second drill will open a read-only copy of the target file, with a number appended to the file name. If you set this to On, then the target file is reused each time you initiate a drill. If you drill a report, leave the target file open, and then drill the report again, the target file is automatically closed and then reopened for the new drill. When this option is enabled, you can only ever have one copy of the drill target open, displaying the results of the most recent drill. **NOTE:** The behavior of this setting is slightly different when the drill source and drill target are the same file. For more information, see Using the same file as the drill source and drill target.

Drill Options

The following settings are available for each drill option. A drill option defines the target report for the drill, and the parameters to pass to the target report. Each enabled drill option defined for a sheet will be available from the **Drill** menu in the **File Options** group.

Field	Description
Name	The name of the drill. This name displays on the **Drill** menu when the drill is activated.
Activate	Specifies whether the drill is active. If **On**, users can select the drill from the Drill menu. If **Off**, the drill does not display on the **Drill** menu. If no drills are active for a sheet, then the **Drill** menu displays "No active drills found on worksheet *name*".

Target File

This section specifies the target file for the drill option. The file must be a report file and must be located in the Reports Library. The user executing the drill must have at least read-only access rights to the target file, as defined by the user's file permissions in Security.

For more information, see Creating the drill target report for custom drilling.

Field	Description
Report folder	The folder where the target file is located. The Reports Library in the Axiom file system is assumed as the location, so only the folder name and any sub-folder names must be specified. For example: • `Budget Reports` if the report is located in the Budget Reports folder • `Budget Reports\Drill` if the report is located in the Drill sub-folder of the Budget Reports folder
Report file to Open	The name of the target file. The full file name must be entered, including the extension. The file must be an Axiom report file.
Sheet to activate	The name of the sheet that you want to be active when the file is opened. This is typically the sheet in the file where the drilling information is being passed from the source file, but it does not have to be.

Field	Description
Target not found error message (optional)	A custom message to display if the target file cannot be found. This is typically used to display a custom message when the user attempts to drill on a row that is not intended to be drilled. You would accomplish this by using a formula in the **Report file to Open** setting, so that the formula resolves to an invalid file name when the user is not on a row that is valid for drilling. **NOTE:** If the **Report file to Open** setting is blank, then no error displays and no action is taken when a user attempts to drill. The setting must resolve to an invalid file name in order to display this message.

Parameters

This section specifies the parameters for the drill option. Typically a drill passes at least two parameters to the output file—a parameter to define the filter for the drill, and a parameter to define the dimensionality for the drill. You can define additional parameters if necessary.

Each parameter can have one or more values. If a parameter has more than one value, then users will be prompted to select the desired value when the drill is executed.

Each parameter has the following settings:

Field	Description
Parameter name	Name of the parameter. If the parameter has multiple values, this name also displays in the dialog where users select a value when drilling.
Cell address for chosen option	Defines the cell address, in the target output file, where you want the parameter value to be passed. The cell address must include the sheet name. For example: `Income_Statement_Drill!C7`. For example, if the parameter is for a filter, this specifies the cell in the target file where the filter will be placed. The target output file would be configured so that this cell is used to define the data in the report (for example as the data filter for an Axiom query).

Field	Description
Value	The value for the parameter. This is the actual value to be passed to the cell address in the target file. The value cell can contain a cell reference or a formula to derive the value based on the user's location in the report, or it can contain a literal value. Literal values are usually only used when the parameter has multiple values for a user to select. If the parameter is for a filter, the cell would likely contain a cell reference to a work area where the user's current context is derived. For example, if the user initiates the drill by double-clicking on a row that contains data for benefit accounts (ACCT.SummaryGroup='Benefits'), the work area would use functions such as Axiom's GetRowNumber and Excel's Index to derive the current row and obtain the criteria statement for the data in that row. The sample formula in row 4 of the Drilling Control Sheet can be adapted for this purpose. If the parameter is for dimensionality, the cell would likely contain a literal value for the user to select. For example, the parameter could have multiple values such as Dept, Dept.VP, and Dept.Region. When the user double-clicks on a row to initiate the drill, they can choose at which level to view the resulting data. If you need more value fields for a parameter, you can add more. See the following section for more details.

Adding more drill options or parameters

By default, the Drilling Control Sheet contains settings for three drill options. If you need more drill options, you can copy and paste an existing section.

1. Unfreeze panes on the Drilling Control Sheet so that you can see the drill tags in Column A.
2. Select all of the rows of an existing drill option, and then copy and paste them below the last drill option in the sheet. You will probably want to leave at least one blank row in between drill options, for ease of reading and reviewing the options.
3. In the first row of the new (pasted) drill option, in column A, edit the `[StartDrill#]` tag to reflect the next sequential number. For example, if the last drill option in the sheet (before you copied) is `[StartDrill3]`, edit the tag of the new option to be `[StartDrill4]`. You should also edit the title in column C to match (i.e. "Drill Option #4").

You can now define the settings for the new drill option as appropriate.

The same concepts apply if you need more drilling parameters or parameter values within an existing drill option. Each parameter has a tag of `[DrillStartParameter#]` (where # is the number of the parameter within the drill option). You can copy and paste an existing parameter section to create a new one, and update the number in the tag.

Each parameter can have multiple parameter values, with a tag of `[DrillParameterValue]`. You can copy an existing value row within a parameter section.

Custom drilling example

This example is provided to help illustrate how custom drilling is set up and how end users interact with it. For this example, we want the custom drill to do the following:

- Pass the filter criteria for the currently selected row in the source report, to be applied as a filter in the target report.

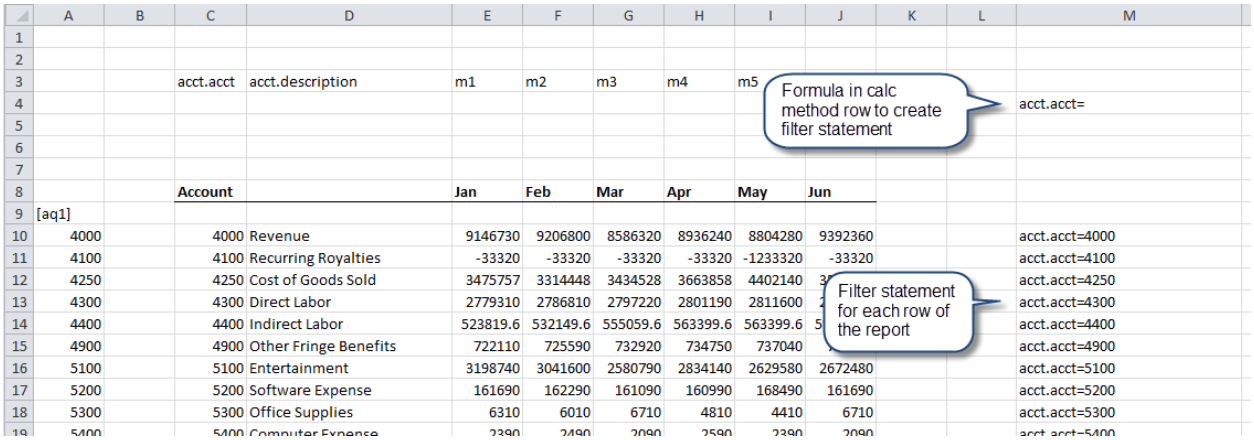
  OR

- Allow the user to select a dimension or grouping to show the drill results at that level.

NOTE: In this particular case, the same end result could be achieved using the built-in drill-down features. However the goal of this example is not to show a typical use case for custom drilling, but rather to illustrate how to set up custom drilling to pass values from the source file to the output file.

Example report to be drilled (the source report)

This sample report uses an Axiom query to return rows by account. The in-sheet calc method in row 4 creates a filter criteria statement for each row (for example, ACCT.ACCT=4000 in row 10).



	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2													
3			acct.acct	acct.description	m1	m2	m3	m4	m5				
4													acct.acct=
5													
6													
7													
8			Account		Jan	Feb	Mar	Apr	May	Jun			
9	[aq1]												
10	4000		4000	Revenue	9146730	9206800	8586320	8936240	8804280	9392360			acct.acct=4000
11	4100		4100	Recurring Royalties	-33320	-33320	-33320	-33320	-1233320	-33320			acct.acct=4100
12	4250		4250	Cost of Goods Sold	3475757	3314448	3434528	3663858	4402140	3			acct.acct=4250
13	4300		4300	Direct Labor	2779310	2786810	2797220	2801190	2811600				acct.acct=4300
14	4400		4400	Indirect Labor	523819.6	532149.6	555059.6	563399.6	563399.6	5			acct.acct=4400
15	4900		4900	Other Fringe Benefits	722110	725590	732920	734750	737040				acct.acct=4900
16	5100		5100	Entertainment	3198740	3041600	2580790	2834140	2629580	2672480			acct.acct=5100
17	5200		5200	Software Expense	161690	162290	161090	160990	168490	161690			acct.acct=5200
18	5300		5300	Office Supplies	6310	6010	6710	4810	4410	6710			acct.acct=5300
19	5400		5400	Computer Expense	2390	2490	2090	2590	2390	2090			acct.acct=5400

For drilling, we want to be able to pass a filter criteria statement to the target file based on the user's current row. To do this, we can use:

- Axiom's GetRowNumber("Active") function to identify the current row.
- Excel's INDEX and INDIRECT functions to return the filter for the selected row.

The Drilling Control Sheet contains a sample function that you can use for this purpose. This function is located in row 4. By default, the function looks as follows:

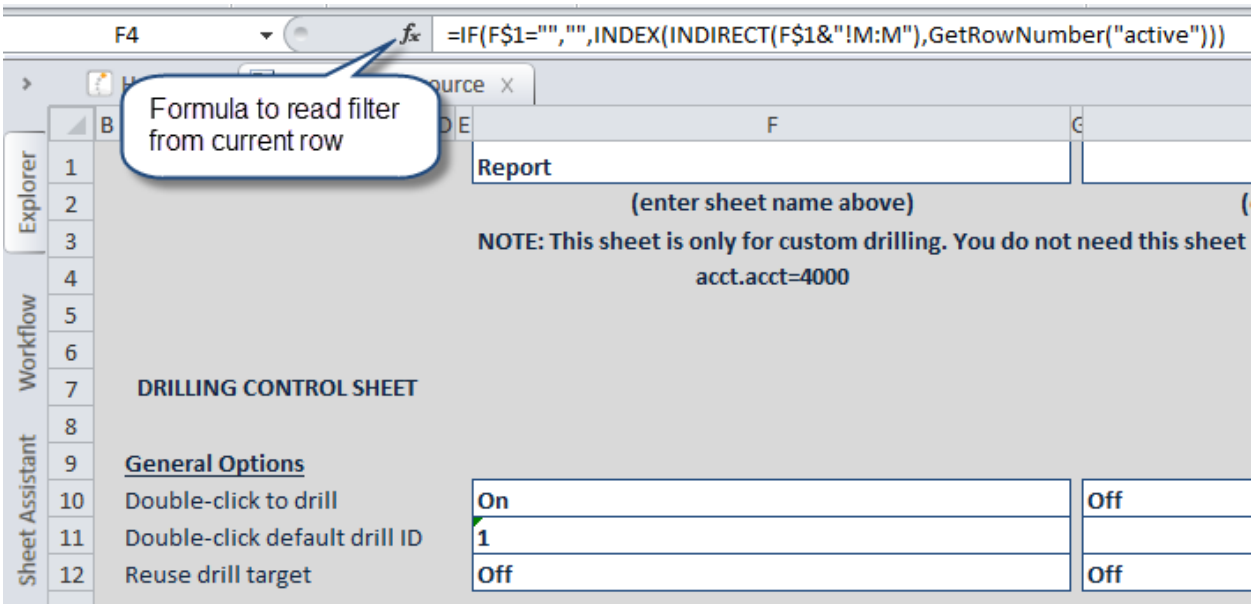
```
=IF(F$1="","",INDEX(INDIRECT(F$1&"!A:A"),GetRowNumber("active")))
```

- The function uses an IF statement so that if no sheet name is entered into row 1 of the Drilling

Control Sheet (in this example, checking cell F1), the function returns blank.

- Otherwise, GetRowNumber("Active") is used to identify the current row. The INDEX and INDIRECT functions are used to look up the current contents of that row in column A, on the sheet to be drilled.

In this case, the only adjustment that we need to make to this formula is to change the column A references to column M, because column M is where we have defined the filter criteria statement for each row.



NOTE: You are not required to use this formula in row 4 of the Drilling Control Sheet. It is provided to give you a starting point to set up context-sensitive drilling. However, you can use any spreadsheet logic that you want in order to set up your custom drilling.

Configuring the drilling parameters

As discussed in the previous section, we have modified the formula in row 4 of the Drilling Control Sheet to return the filter for the current row. Now, we need to set up the drilling parameters to pass this filter to the target file, and also to pass the desired grouping level to the target file.

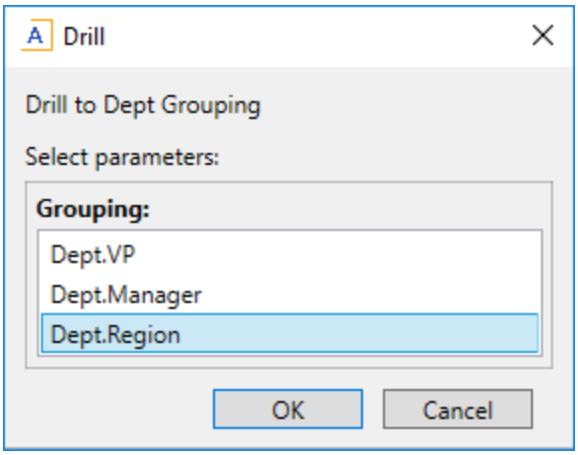
In this example, Parameter #1 is used to pass the drill filter. The Value cell points to the formula in row 4 to get the current filter. Drill!C5 is where we want to place the filter in the output file.

Parameter #2 is used to pass the desired grouping level. In this case the Value cells are hard-coded values, and the user will select the value that they want. Drill!C6 is where we want to place the grouping level in the output file.

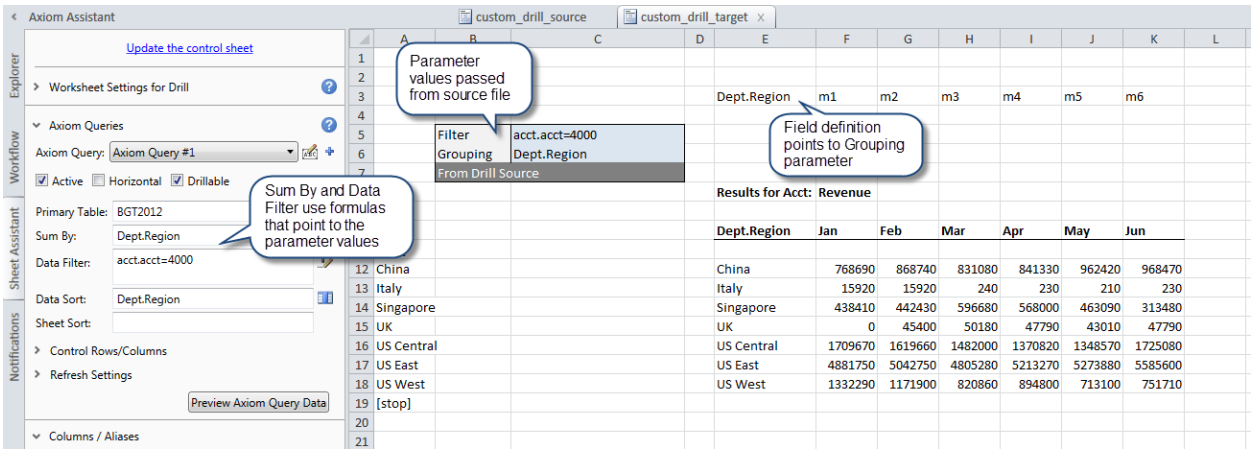
23	**Parameters**	
24	**Parameter #1**	
25	Parameter name	**Filter**
26	Cell address for chosen option	**Drill!C5**
27	Value	**acct.acct=4000**
28		
29	**Parameter #2**	
30	Parameter name	**Grouping**
31	Cell address for chosen option	**Drill!C6**
32	Value	**Dept.VP**
33	Value	**Dept.Manager**
34	Value	**Dept.Region**

Example drill results (the target report)

When the user initiates a drill (either by double-clicking a row or using the Drill menu), the Drill dialog presents the user with the list of options from Parameter #2. The user is not prompted to do anything with Parameter #1 because there is only one value to be passed and therefore it is used by default.



The default filter value (for example: ACCT.ACCT=4000) and the selected grouping value (for example: DEPT.Region) are then passed to the target cell addresses on the drill target file. These values can then be used in the Field Definition Row of an Axiom Query, and for Axiom Query settings on the Control_Sheet of the drill target file.



With these parameters the drill target file now contains the selected grouping and the drill filter for the row the user drilled on. Since the drill target file is another Axiom report, the format and structure can easily be customized to return data from the same table as the original report, or from another table or tables that relate to the selected row.

Designating non-drillable rows for custom drilling

When using custom drilling, Axiom considers all rows as eligible for drilling by default. This is because there is no predefined drill context—what is eligible for drilling is entirely up to you and how you have configured the drill. This allows for maximum flexibility in setting up the drill, but it also means that by default users can drill on any row in the sheet—empty rows, title rows, etc.—and the target drill file will still open and attempt to apply the drill parameters.

If you want to prevent drilling on invalid rows, you can use formulas to adjust certain drilling settings. These formulas would employ the same logic that you are using to determine the active row for the drill and gather information from the active row. The following options can be used to handle invalid rows.

- You can set up a formula in the **Activate** setting for the drill, so that the setting changes to **Off** if a user is on an invalid row. In this case no message will display to the user when they attempt to drill, the drill will simply not happen when the user is on that row.
- You can set up a formula in the **Report file to Open** setting for the drill, so that the value of this setting changes if the user is on an invalid row.
    - If the formula returns blank, this has the same effect as disabling **Activate** for the drill.
    - If the formula returns an invalid file name, you can display a custom message to the user, as defined in the setting **Target not found error message**. For example, you can display a message such as: "This is not a valid row for drilling."

The first two options should only be used if the primary means of drilling is by using double-click. In this case the drill context is determined when the user double-clicks, and then the drill will be enabled or not as appropriate. If the drill is disabled based on the current context, then nothing will happen. However, the Drill menu on the Axiom ribbon may not display correctly, because it is displaying the drill options based on the last calculation in the sheet. The drill will still be disabled if the user is on an invalid row, but

it may be confusing to the user to see the drill option on the menu but nothing happens when it is clicked. If you want to cover all drilling use cases, the best option is to use the invalid target file name with the custom error message.

For example, imagine that your custom drill is determining the active row and reading a filter criteria statement from that row, and then passing the filter to the target file for the drill. You can set up the formula so that if the column that normally contains the filter criteria statement is instead blank on a certain row, then the target file becomes set to an invalid file name and the custom message is displayed to the user.

11

Action Code Processing

Axiom files can be set up with *action codes* that trigger copy, lock, and unlock actions when the sheet is updated with data.

Setting up action code processing for Axiom files

You can use action codes to automatically perform certain actions in a sheet when the sheet is updated with data. These codes can be used to lock and unlock cells, and to copy cell contents from one location to another.

In general, action codes are processed after Axiom queries are run, and after calc methods are inserted or changed in a sheet. Some additional processing options are also available using special commands.

Action code processing depends on the placement of reserved action tags in the sheet. There are two components:

- The ActionCodes tag, which defines a control column and a control row for the action code processing.
- Matching action tags to specify "action cells" for processing. Each intersection of matching action tags in the control column and the control row specifies an action cell for processing. The paired tags specify the action to perform.

You can have lock, unlock, and copy pairs within the same action control row and control column. You can also combine lock, unlock, and copy tags, by delimiting the tags with a comma.

Action code processing is a flexible feature that can be set up in a variety of ways. See the detailed topics in this section for more details on setting up each type of operation.

NOTE: Action code processing can be resource-intensive. When designing action codes in a file, care should be taken to minimize the performance impact of this process. The number of actions and the frequency of their processing should be limited to only processing when absolutely necessary.

Defining the ActionCodes tag

In order to use action code processing, you must place an ActionCodes primary tag within the sheet. Within the ActionCodes control row and control column, you can define copy, lock, and unlock actions.

To define the location of the control row and control column for action code processing, place the following tag in any cell in the sheet, within the first 500 rows:

```
[ActionCodes]
```

The column that contains the tag becomes the control column, and the row that contains the tag becomes the control row. Action tags can be placed anywhere in relation to this tag, within the control row and control column. For example, control row codes can be placed to the right or to the left of the ActionCodes tag, and control column codes can be placed underneath or above the ActionCodes tag.

The control row and control column can contain other information that does not pertain to the action codes; they do not have to be reserved for action codes only. The action code processing will ignore any content in the control row or control column that does not begin with any of the reserved action tags.

NOTES:

- The primary ActionCodes tag must be located in the first 500 rows of the sheet.
- The ActionCodes tag can be placed within a formula, as long as the starting bracket and identifying tag are present as a whole within the formula. For more information, see Using formulas with Axiom feature tags.

Using multiple ActionCodes tags

You can have multiple ActionCodes tags in the same sheet, and they will all be processed when one of the triggering events occurs.

If you have more than one ActionCodes tag in a sheet, it is possible to create situations where action pairs from one ActionCodes control intersect on the same action cell as pairs from another ActionCodes control. Make sure to check for this when testing the action code implementation. If necessary, you can use named actions to work around this (see Using named action tag pairs).

Using a named ActionCodes tag

You can name an ActionCodes tag so that it can be explicitly run using manual processing commands. The syntax for a named tag is as follows:

```
[ActionCodes:Name]
```

IMPORTANT: Named ActionCodes tags are ignored during normal action code processing (i.e. after Axiom query refresh and during calc method insertion or change). For more information, see How action code processing executes.

The typical use case for named tags and manual processing is in Axiom forms. Using a button command, users can trigger processing of a particular named tag at a particular point in the form processing.

Setting up copy actions

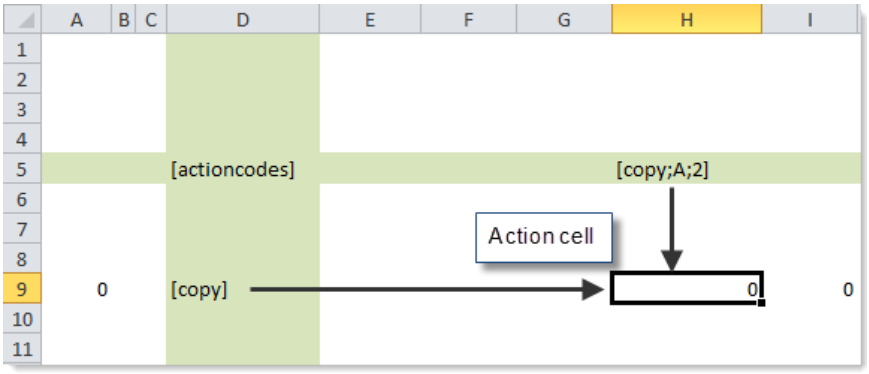
Using action code processing, you can copy cell formatting and/or content from one location to another within the sheet. The Copy tag is used to perform the copy action, and this tag defines the following:

- The source location to be copied
- The target location for copying
- The scope of the copy action: formulas or values only, formats only, or both

Use of copy actions requires an ActionCodes tag to be set up in the sheet. Within the ActionCodes control column and control row, you can define copy, lock, and unlock actions.

Creating action tag pairs for copy actions

To define a copy action, you create an *action tag pair* by placing corresponding copy tags in the ActionCodes control row and control column. The intersection of the action tag pair defines an *action cell* for processing. For example:



Each action tag pair must consist of the following:

- One copy tag that contains the necessary parameters to define the copy action. At minimum, a source location must be specified for the copy action. The other parameters may be omitted to assume default actions.
- One copy tag without parameters (simply `[Copy]`). The purpose of this tag is to create the action tag intersection that defines the action cell.

In this example, the copy tag that defines the copy action is placed in the ActionCodes control row, and the corresponding "simple" copy tag is placed in the ActionCodes control column. You can reverse the placement of the tags if desired. As long as the action tag pair is valid, it doesn't matter where each tag is placed.

If you want the copy operation to apply when inserting or overwriting calc methods, the calc method itself must be saved with the necessary action tag for the row (in the control column). That way, when the calc method is brought into the sheet, the control column will contain an action tag, and a copy operation can be applied to the calc method.

If an action tag pair is invalid, or if a tag contains invalid syntax, then the action code processing is stopped before any actions are taken and an error message is displayed (or logged, if processing as part of a utility). For example, if an intersection occurs between two copy definition tags (for example, if `[Copy;A1]` intersects with `[Copy;B;2]`), then that is an invalid action tag pair.

NOTES:

- Only matching action tag pairs are validated. If a copy tag intersects with a lock tag, those tags are not considered an action tag pair and are ignored. However, each individual tag is validated for correct syntax.
- For each ActionCodes tag, we recommend placing all of your copy definition tags in either the control row or the control column (so that all the copy definition tags are in one location, and all of the corresponding "simple" tags are in the other location). If you alternate placement of the tags (with some definition tags in the control row, and some definition tags in the control column), you run a greater risk of accidentally creating invalid action tag pairs. Another way around this is to use named copy tags, so that intersecting tags are only considered an action tag pair if they have the same name. See Using named action tag pairs.

Copy tag syntax

Copy tags use the following syntax:

`[Copy;SourceLocation;TargetSize;PasteType]`

The Copy part of the tag identifies it as a copy action and must be included. The other parameters may be defined as follows:

Parameter	Description
SourceLocation	The cell or cells to be copied. The source location can be one of the following: • An absolute cell address, such as A1. • An offset location, relative to the action cell. By default, the source location is assumed to be a single cell. If you want to copy a range of cells, you can specify an optional size for the source location, in the format *W*x*H* (width by height). If used, the size is appended to the source location using a colon. For example: `A1:2x2` copies a block of cells that is two columns wide and two rows high, starting at cell A1. The source location must be specified on one (and only one) of the copy tags in an action tag pair. If you want the source location to be the action cell itself, this must be specified as an offset (+0). For more information on defining the source location, including the syntax for specifying an offset location, see the *Source location remarks and examples* following this table.
TargetSize	Optional. The area to copy into, in the format *W*x*H* (width by height), in relation to the action cell. You should omit the target size if you want to assume the copy area based on the specified source size. For example, if the source is one cell, then the contents of that single cell will be copied into the action cell. If the source is two cells wide, those two cells will automatically be copied, starting at the action cell. The target size is used when you want to copy a single source cell into multiple target cells. For example, you could specify a target size of 3x1, and the source cell will be copied into the action cell and the two cells immediately to the right of the action cell (the three-cell range starting with the action cell). The target size always begins with the action cell and then progresses to the right (the width of the area) and down (the height of the area). **NOTE:** You can specify either a source size or a target size, but not both. Since the target size is assumed automatically if omitted, there is no reason to specify it if a source size is specified.

Parameter	Description
PasteType	Optional. Specifies what to paste into the target cells: • `Formats`: Copies formats only. • `Formulas`: Copies formulas and number formats. • `FormulasOnly`: Copies formulas only. • `Values` : Copies values only. • `All` : Copies cell contents and formats. If omitted, the default is `All`.

NOTES:

- When specifying a size (*W*x*H*), you can omit the height if you want to use the default height of 1. For example, if you want to copy an area that is three cells wide, you can specify `3` instead of `3x1`. However, if the width is 1 and you want to specify a height, the width must be included: `1x3`. If only one number is specified as the size, it is always interpreted as the width.
- Action tags can be placed within a formula if desired. For example, you might use a formula to determine the width of the target size based on the current period, or you might use an IF formula to determine whether the cell contains an action tag or not.

Source location remarks and examples

When defining the source location for the copy action, you can specify an absolute cell reference or an offset location.

Absolute cell references can be to any cell in the workbook: for example, `A1` or `Sheet1!A1`. You do not have to place dollar signs ($) in the cell reference to make it absolute—A1 is always interpreted as A1, regardless of where in the sheet the action is being performed.

The offset location is relative to the action cell, and can be specified as follows:

- **Relative column and relative row**: Both the column and row are relative to the action cell. The relative location is specified as follows:

  +C#+R#

  Where *C#* is the number for the column offset and *R#* is the number for the row offset.

- **Absolute column and relative row**: The column is absolute, but the row is relative to the action cell. The location is specified as follows:

  CL+R#

  Where *CL* is the column letter and *R#* is the number for the row offset.

When specifying an offset, the number can be positive or negative. Positive numbers are evaluated to the right and down from the action cell, and negative numbers are evaluated to the left and up from the action cell.

The default value for the row offset is zero (meaning, the row containing the action cell) and can be omitted. The column must always be specified, either as an offset or as an absolute column.

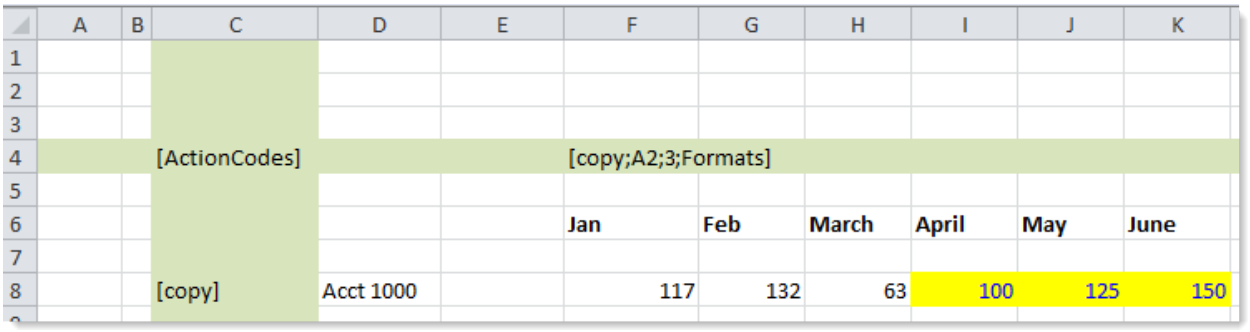
The following are examples of valid source locations:

Source Location	Type	Description
A1	Absolute	Copy cell A1 in the current sheet.
A1:3x2	Absolute	Copy a block that is 3 cells wide and 2 rows high, starting at cell A1.
A1:3	Absolute	Copy a block that is 3 cells wide, starting at cell A1.
-12	Relative / Relative	Copy the cell that is 12 columns to the left of the action cell, within the same row as the action cell.
-12+2	Relative / Relative	Copy the cell that is 12 columns to the left of the action cell and 2 rows down from the action cell.
+8-2:6	Relative / Relative	Copy a block that is 6 cells wide, starting at the cell that is 8 columns to the right and 2 rows up from the action cell.
B	Absolute / Relative	Copy the cell in column B, within the same row as the action cell.
B+2	Absolute / Relative	Copy the cell that is in column B, two rows down from the action cell.
B-2:1x2	Absolute / Relative	Copy a block that is 1 cell wide and 2 rows high, starting at the cell that is in column B and two rows up from the action cell.
+0	Relative / Relative	Copy the contents of the action cell. For example, the action cell might contain a formula, the result of which you want to copy and paste as a value.

Copy tag examples

A typical use for copy actions is when performing rolling forecasting. Each month, the plan files would be refreshed to bring in the latest month of actuals. As the actuals are brought in, copy actions are used to change the formatting of the cells from the yellow-background, blue-text format used for input cells, to the white-background, black-text format used for actual values from the database.

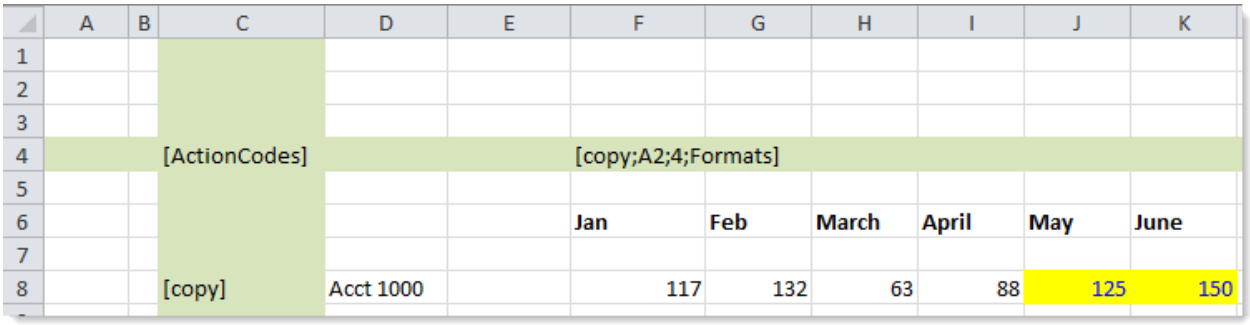
For example, the following screenshot shows how the row would appear when the current planning period is period three. (This is not an actual implementation example; it is intended to illustrate the copy concepts.)



	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4			[ActionCodes]			[copy;A2;3;Formats]					
5											
6						**Jan**	**Feb**	**March**	**April**	**May**	**June**
7											
8			[copy]	Acct 1000		117	132	63	100	125	150

The values in Jan-March are actual values from the database, while the remaining months have inputted projection values.

When the current planning period is changed to period 4 and the plan files are refreshed, actuals are brought in for April and the copy operation is extended to affect the column for that month:



	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4			[ActionCodes]			[copy;A2;4;Formats]					
5											
6						**Jan**	**Feb**	**March**	**April**	**May**	**June**
7											
8			[copy]	Acct 1000		117	132	63	88	125	150

Note the following about this example:

- A formula is being used in the copy tag to retrieve the current planning period value and set the target size as appropriate. Previously the formula returned 3, now it returns 4. Therefore the copy operation is applied to the four-cell block starting at the action cell of F8.
- The designated source location of A2 is providing the white-background, black-text format for the copy operation. If the source cell is locked, the target cell will also be locked (when copying formats), so we do not also need a Lock action in this case.
- Only formats are being applied in this case, as specified in the last parameter of the copy tag.
- In this scenario, the copy operation must apply to the full range of months, not just April, so that it can also handle the case of inserting new calc methods. If a new calc method is inserted at this point in time, the copy operation would be applied to months Jan-April, and the user can begin inputting forecasting values for the month of May.

The following are some additional examples of copy tags:

`[Copy;A10;Formats]`

This example copies the format from A10 into the action cell. Note that when the optional target size parameter is omitted, it does not have to be delimited with an empty semicolon. However, both constructions are valid (the syntax `[Copy;A10;;Formats]` will also work).

`[Copy;+8-2]`

This example copies the cell that is 8 columns to the right and two rows up from the action cell. Because the paste type is omitted, it is assumed to be All, which means the contents of the cell will be copied "as is", including formats.

`[Copy;B:1x3;Formulas]`

This example copies formulas from a three-cell-high block that starts in the cell that is located in column B, in the same row that contains the action cell. These formulas are pasted into a three-cell-high block starting at the action cell. Note that number formats are copied along with the formulas. If you do not want the number formats to be copied, use FormulasOnly instead.

`[Copy;B;3;Values]`

This example copies values from the cell in column B, in the same row as the action cell. The values are pasted into a three-cell wide block starting with the action cell.

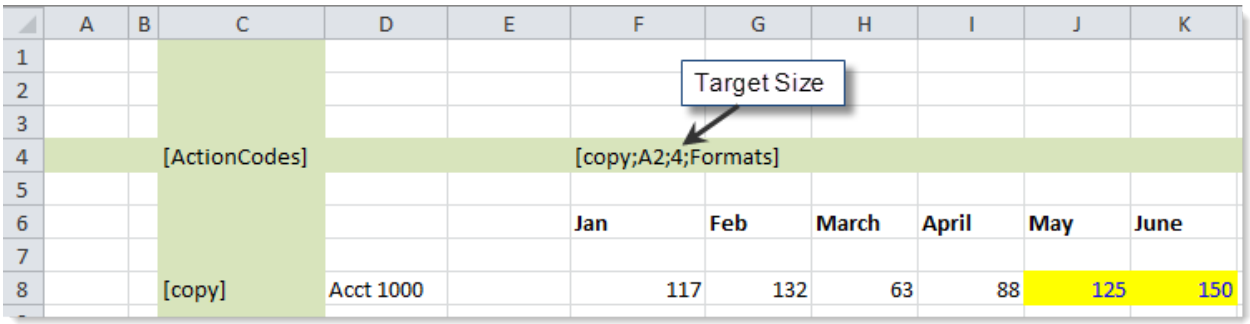
`[Copy;+0;Values]`

In this example, the action cell itself is the source location (offset of +0). For example, the action cell might contain a formula, and you want to take the result of the formula and paste it back into the action cell as a value.

Performance considerations for copy actions

When possible, it is best to specify a size for either the source or the target to copy a block of cells at once, rather than having several individual copy operations. This minimizes the number of action cells to be processed and can accomplish the same end result with fewer copy operations.

The example in the previous section specified a size for the target. As the current planning period changes, the size increases to cover the necessary number of cells:



Example 1

You could achieve the same effect without specifying a size. The following example uses formulas in the control row to determine when each individual column should be marked with `[Copy]`. As the current planning period changes, a new column would become flagged for a copy operation.

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4			[ActionCodes]			[Copy]	[Copy]	[Copy]	[Copy]		
5											
6						**Jan**	**Feb**	**March**	**April**	**May**	**June**
7											
8			[copy;A2;;Formats]	Acct 1000		117	132	63	88	125	150

Example 2

While both examples accomplish the same end result, the first example is preferred because it has one action cell to evaluate and one copy action to perform. The format is pasted into all four cells in one operation.

The second example has four action cells to evaluate and four separate copy actions to perform. While the performance difference is likely not noticeable for a single row, when multiplied across many rows in the sheet (and perhaps additional copy or lock operations) the first example should result in faster performance.

TIP: In general, spreadsheet processing in the Windows Client is faster than in the Excel Client. If the processing speed for action codes seems slow in the Excel Client, try using the Windows Client to run the Process Plan Files utility (either manually or via Scheduler) or to refresh individual plan files.

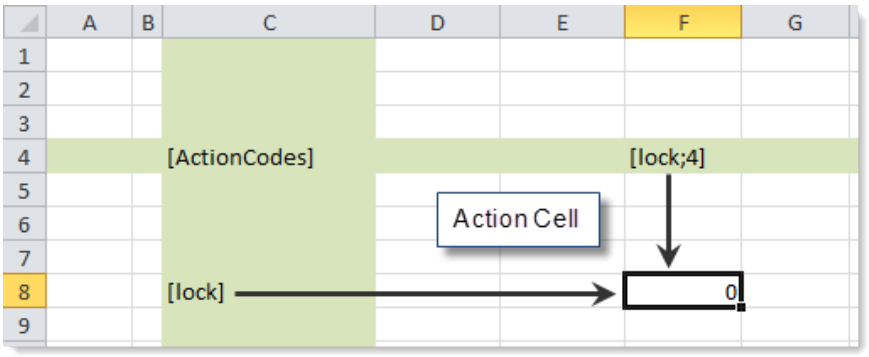
Setting up lock and unlock actions

Using action code processing, you can lock or unlock cells in the sheet. The Lock tag is used to lock cells, and the Unlock tag is used to unlock cells. If sheet protection is used on the sheet, locked cells cannot be edited, and unlocked cells can.

Use of lock and unlock actions requires an ActionCodes tag to be set up in the sheet. Within the ActionCodes control column and control row, you can define copy, lock, and unlock actions. For more information, see Defining the ActionCodes tag.

Creating action tag pairs for lock or unlock actions

To define a lock or unlock action, you create an *action tag pair* by placing corresponding lock or unlock tags in the ActionCodes control row and control column. The intersection of the action tag pair defines an *action cell* for processing. For example:



The lock and unlock tags support one optional parameter, to define a target size for the lock or unlock operation. Within an action tag pair, one tag can have the optional parameter, and the other tag must be just `[Lock]` or just `[Unlock]`.

In this example, the lock tag with the size parameter is placed in the ActionCodes control row, and the corresponding "simple" lock tag is placed in the ActionCodes control column. You can reverse the placement of the tags if desired. As long as the action tag pair is valid, it doesn't matter where each tag is placed. If you are not using the optional parameter, then the lock and unlock pairs are exactly the same.

If you want the lock or unlock operation to apply when inserting or overwriting calc methods, the calc method itself must be saved with the necessary action tag for the control column. That way, when the calc method is brought into the sheet, the control column will contain an action tag, and a lock or unlock operation can be applied to the calc method.

If an action tag pair is invalid, or if a tag contains invalid syntax, then the action code processing is stopped before any actions are taken and an error message is displayed (or logged, if processing as part of a utility). For lock and unlock tags, the only invalid combination is if you have two intersecting tags that both use the optional size parameter. For example, if `[Lock;4]` in the control row intersects with `[Lock;2]` in the control column, that is an invalid action tag pair. However, if you have a lock tag that intersects with an unlock tag (or vice versa), that combination is ignored and no error occurs.

NOTE: Only matching action tag pairs are validated. If a copy tag intersects with a lock tag, those tags are not considered an action tag pair and are ignored. However, each individual tag is validated for correct syntax.

Lock and Unlock tag syntax

Lock and unlock tags use the following syntax:

```
[Lock;Size]

[Unlock;Size]
```

In both cases, size is an optional parameter that determines the size of the area to lock or unlock. The size is specified using the format *W*x*H* (width by height), in relation to the action cell. If omitted, only the action cell is locked or unlocked.

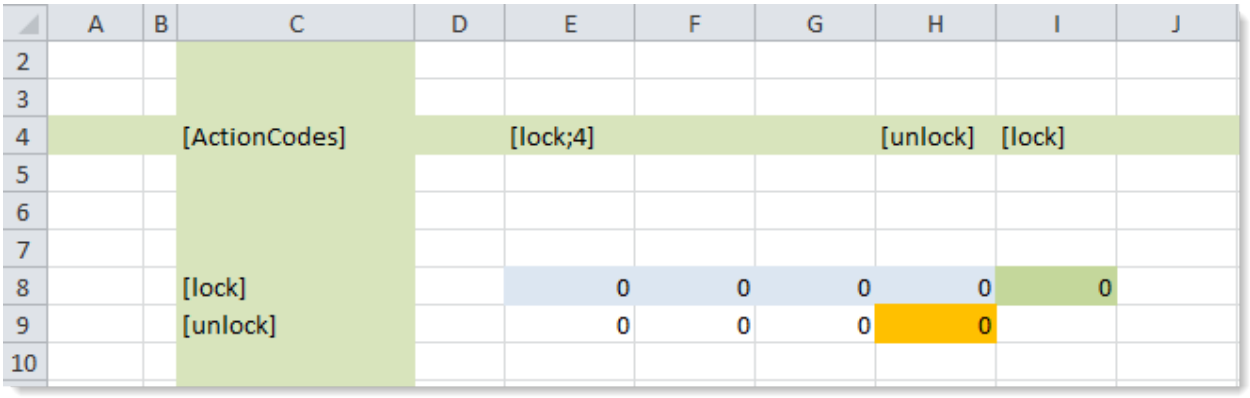
If specified, the area to lock or unlock is calculated starting at the action cell. For example, if the size is specified as `3x2`, that is an area three columns wide by two cells high, starting at the action cell. All of the cells in that area are locked or unlocked.

When specifying a size, you can omit the height if you want to use the default height of 1. For example, if you want lock an area that is three cells wide, you can specify `3` instead of `3x1`. However, if the width is 1 and you want to specify a height, the width must be included: `1x3`. If only one number is specified as the size, it is always interpreted as the width.

NOTE: Action tags can be placed within a formula if desired. For example, you might use an IF formula to determine whether the cell contains an action tag or not.

Lock and Unlock tag example

The following example shows several lock and unlock actions. This is not a real-life implementation example; it is intended to illustrate the lock and unlock concepts.



	A	B	C	D	E	F	G	H	I	J
2										
3										
4			[ActionCodes]		[lock;4]			[unlock]	[lock]	
5										
6										
7										
8			[lock]		0	0	0	0	0	
9			[unlock]		0	0	0	0		
10										

- The first action cell is E8. Because the lock tag in the control row uses the size parameter, the four blue cells will be locked.
- The next action cell is I8. The size parameter is not used here, so only the green cell will be locked.
- The next action cell is H9. The size parameter is not used here, so only the orange cell will be unlocked.

The intersections of lock and unlock (in H8, E9, and I9) are ignored. Only pairs of lock / lock and unlock / unlock are considered to be action tag pairs.

Combining action tags

When using action code processing, you can combine Copy, Lock, and Unlock tags, so that multiple actions can be performed using the same set of codes.

To combine action tags, separate the tags with a comma, within a single set of brackets. The following example combines a copy tag with a lock tag:

```
[Copy;A1;Formulas,Lock]
```

This example copies the formula in A1 into the action cell, and then locks the action cell. The corresponding "simple" tag to create an action tag pair would be `[Copy,Lock]`.

NOTES:

- The order of the combined tags does not matter. A tag of `[Lock,Copy]` is interpreted in the same way as `[Copy,Lock]`.
- You only need to combine a copy tag with a lock tag if you are not copying formats. If you are copying formats, then the source cell can be locked or unlocked as desired, and this format will be copied to the target cell. However, if you are copying only formulas or values, then you would use a combined copy and lock tag if you want the target cell to be locked.

You can combine any number of action tags into a single tag. If you are using named actions, you can have multiple instances of the same type of action within the same tag. For more information on named actions, see Using named action tag pairs.

You can combine lock and unlock codes in the same tag, so that the same tag can be used for different lock and unlock action tag pairs. In the following example, the tag in the control column has a combined lock and unlock tag, so that it can form a valid action tag pair with both of the lock and unlock tags in the control row. The blue cells in E8:H8 will be locked, and the orange cell in I8 will be unlocked.

	A	B	C	D	E	F	G	H	I
2									
3									
4			[ActionCodes]		[lock;4]				[unlock]
5									
6									
7									
8			[lock, unlock]		0	0	0	0	0
9									

Combined lock / unlock example

If the tag in C8 was just `[Lock]`, then it would not match with the unlock tag in cell I4, and no action would be performed on I8.

Using named action tag pairs

When using action code processing, you can define names for action tags to create explicit matching pairs of tags. You may wish to name action tags if there are many action tags in the sheet, and you want to ensure that only certain combinations of those tags match up within the sheet.

To define a name for an action tag, append the name to the tag using a colon. For example:

```
[Copy:Actuals;A1;3;Formats]
```

This tag now *only* matches with other tags that contain the text `[Copy:Actuals]`. If the tag intersects with a regular copy tag (`[Copy]`), or with a copy tag with a different name (`[Copy:Budget]`), that intersection is not considered an action tag pair and is ignored.

NOTE: The purpose of named action tag pairs is different than named ActionCodes tags. Named action tag pairs provide a way to prevent accidental matches within the sheet. Named ActionCodes tags provide a way to process only the action codes within that particular ActionCodes control row and control column, on demand. For more information, see Defining the ActionCodes tag.

In the following screenshot, the only action cells are F8 and N9. Because the tag in F4 contains the name "Actuals," it only matches with the corresponding tag in C8. F9 and N8 are not action cells, because their tag names do not match.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3															
4			[ActionCodes]			[Copy:Actuals;A2;4;Formats]								[Copy;+20+2;Formulas]	
5															
6						Jan	Feb	March	April	May	June				
7															
8			[Copy:Actuals]	Acct 1000		117	132	63	88	125	150				
9			[Copy]												
10															

If you had a multi-row calc method where you needed to perform a different copy action in the same column but in a different row, you could name the second copy action and combine the action tags like so:

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4			[ActionCodes]			[Copy:Actuals;A2;4;Formats,Copy:Calc;+28+2;Formulas]					
5											
6						Jan	Feb	March	April	May	June
7											
8			[Copy:Actuals]	Acct 1000		117	132	63	88	125	150
9			[Copy:Calc]								
10											

Now cell F4 contains the named tags for both copy actions. The Copy:Actuals action is performed on cell F8, and the Copy:Calc action is performed on cell F9. In this particular example, you could get around the need to name the actions by moving the copy definitions to the control column instead of the control row, like so:

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4			[ActionCodes]			[Copy]					
5											
6						Jan	Feb	March	April	May	June
7											
8			[Copy;A2;4;Formats]	Acct 1000		117	132	63	88	125	150
9			[Copy;+28+2;Formulas]								
10											

Now the definitions are per row instead of per column, and both match with the "simple" tag in F4. You could decide to use either setup, depending on what other actions you need to accomplish in this sheet.

How action code processing executes

Action code processing can occur automatically when data is updated in a sheet, or "on demand" using special processing commands.

Automatic processing

Unnamed ActionCodes tags are processed automatically whenever one of the following events take place. This automatic processing occurs in all environments, including Axiom forms.

- After refreshing Axiom queries on the sheet.
- After a calc method is inserted or overwritten in the sheet.

NOTES:

- In the case of inserting or overwriting calc methods, only the affected rows are processed.
- If an Axiom query is configured to refresh on open, only the action codes on the same sheet as the refreshed Axiom query will be processed.
- Action codes are never automatically processed in file group templates. Once plan files are created from template, the action codes will be processed in the plan files.

Named ActionCodes tags (for example, `[ActionCodes:MyName]`) are always ignored during this automatic processing. Named tags can only be processed using manual processing.

Manual processing

Both named and unnamed ActionCodes tags can be processed manually using the following features:

- Use the **Process** button on the **Axiom Designer** tab to manually process an individual ActionCodes tag, either named or unnamed. This feature is primarily intended for purposes of testing the action codes setup and not for regular processing. For more information, see Using the Axiom Designer tab. This feature is also available as a command (Process Single Action Code) that can be used in any custom ribbon tab.

  NOTE: This command can be used in file group templates to test action code setup.

- Use the **Process Action Codes** command to trigger the processing of either all unnamed ActionCodes tags, or of a single specific named tag. This command can be used in Axiom forms or in custom task panes and ribbon tabs to provide on-demand action code processing. For more information on custom task panes and ribbon tabs, see the *System Administration Guide*. For more information on Axiom forms, see the *Axiom Forms and Dashboards Guide*.

Order of action code processing

Before any actions are performed, Axiom first identifies all ActionCodes tags that are currently eligible for processing in the sheet, and then identifies all actions associated with those tags. The tags eligible for processing depend on how processing is triggered, as discussed in the previous sections. It may be all unnamed ActionCodes tags, or a single named ActionCodes tag.

Once all actions for all eligible tags are identified, these actions are performed in the following order:

- Copy actions
- Lock actions
- Unlock actions

Action codes quick reference

This topic is a quick reference for action codes. See the detailed topics in this section for more information.

What can action codes do?

Action codes can be used to:

- Copy formats, formulas, and/or values from one location in a sheet to another. For more details on defining copy actions, see Setting up copy actions.
- Lock or unlock cells in a sheet. For more details on defining lock or unlock actions, see Setting up lock and unlock actions.

NOTE: Action code processing can be resource-intensive. When designing action codes in a file, care should be taken to minimize the performance impact of this process. The number of actions and the frequency of their processing should be limited to only processing when absolutely necessary.

What triggers action codes?

Action code processing in a sheet is triggered by the following events:

- **Refreshing Axiom queries.** If only the current sheet is refreshed, then only the action codes on that sheet are processed. If the whole workbook is refreshed, then all action codes are processed. Named ActionCodes tags are ignored.
- **Inserting or overwriting a calc method on the sheet.** Action codes are only processed for the rows impacted by the insertion or change operation. Named ActionCodes tags are ignored.
- **Running the Process Action Codes command.** The behavior depends on whether an ActionCodes tag name is specified. If no name is specified, then all action codes in the workbook are processed (excluding named ActionCodes tags). If a name is specified, then only the named ActionCodes tag is processed.

Administrators can use the **Action Codes** group on the **Axiom Designer** tab to process any set of action codes on demand, without depending on Axiom queries or other means of triggering action code processing. This is intended for testing the action codes setup. See Using the Axiom Designer tab.

How are action codes defined in a sheet?

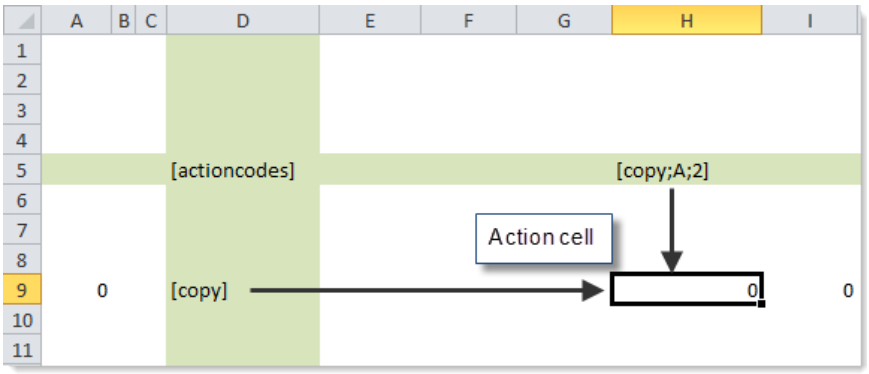
Action code processing depends on the placement of reserved tags in the sheet. There are two components:

- The ActionCodes tag, which defines a control column and a control row for the action code processing.
- Matching action tags to specify "action cells" for processing. Each intersection of matching action tags in the control column and the control row specifies an action cell for processing. The paired tags specify the action to perform.

Action tag summary

Tag Type	Tag Syntax
ActionCodes primary tag	`[ActionCodes]`
Lock action tag pairs	`[Lock;TargetSize]` `[Lock]`
Unlock action tag pairs	`[Unlock;TargetSize]` `[Unlock]`
Copy action tag pairs	`[Copy;SourceLocation;TargetSize;PasteType]` `[Copy]`

Action codes perform the copy, lock, or unlock action on the *action cell*. Action cells are defined by the intersection of matching action tags.



You can have lock, unlock, and copy pairs within the same action control row and control column. You can also do the following:

- Combine lock, unlock, and copy tags within the same cell, by delimiting the tags with a comma. For more details, see Combining action tags.
- Define "named" action tags, so that action tags only "match" with other tags that have the same name. For more details, see Using named action tag pairs.
- Define "named" ActionCodes tags, in order to explicitly process that set of action codes on demand. For more details, see Defining the ActionCodes tag.

Action code examples

The following are examples of action code tags. Only the "primary" tag is shown, meaning, the tag that contains the parameters to define the copy, lock, or unlock action. All tags must have a corresponding "simple" tag to define the action cell.

NOTE: For copy actions, if TargetSize is not being specified, but you want to specify the PasteType, you can simply omit the unused parameter. You do not need to delimit the unused parameter with an "empty" semicolon (although that will also work).

Example	Description
[Copy;A10]	Copies a specific cell (A10) to the action cell.
[Copy;+0;Values]	Copies the contents of the action cell back into the action cell as values. For example, this could be used to convert a formula to a value, within the same cell. The offset +0 is used to specify the action cell as the source location.
[Copy;A10;Formats]	Copies the format of a specific cell (A10) to the action cell.
[Copy;A10;1x3;Formats]	Copies the format of a specific cell (A10) to a three-cell high block starting at the action cell.
[Copy;A10;2x2;Formats]	Copies the format of a specific cell (A10) to a block of cells two columns wide and two rows high, starting at the action cell.
[Copy;+8-2]	Copies a cell that is 8 columns to the right and two rows up from the action cell.
[Copy;B:1x3;Formulas]	Copies formulas and number formats from a three-cell high block that starts in the cell that is located in column B, in the same row that contains the action cell. These formulas are pasted into a three-cell high block starting at the action cell.
[Copy;B;3;Values]	Copies values from the cell in column B, in the same row as the action cell. The values are pasted into a three-cell wide block starting with the action cell.
[Copy;A10:3x2;FormulasOnly]	Copies formulas only from a block of cells three columns wide and two rows tall, that starts at a specific cell (A10). The formulas are copied into an equivalent block of cells that starts at the action cell.
[Copy;B-2:1x2;Formats]	Copies formats from a two-cell high block of cells that starts at the cell in column B and two rows up from the action cell. The formats are copied into an equivalent block of cells that starts that the action cell.
[Lock]	Locks the action cell.
[Lock;4]	Locks a four-cell wide block of cells that starts at the action cell.
[Lock;1x4]	Locks a four-cell high block of cells that starts at the action cell.
[Unlock]	Unlocks the action cell.
[Unlock;4]	Unlocks a four-cell wide block of cells that starts at the action cell.
[Unlock;1x4]	Unlocks a four-cell high block of cells that starts at the action cell.

12

Master Sheets

You may want end users to be able to add sheets to their Axiom files as needed, using designated *master sheets* as the sheet "templates." This feature provides a controlled way for end users to add sheets, even if workbook protection is enabled for the file.

For example, you may want your department managers to add a CapRequest sheet to their plan file for each capital request that they want to submit as part of their budget. Some departments may have zero CapRequest sheets, other departments may have just one CapRequest sheet, and other departments may have two, three, or more CapRequest sheets. The primary reasons to use master sheets in this example are:

- You want each request on a dedicated sheet within a plan file, versus using other approaches such as detail rows on a single sheet, or individual plan files for each request.
- You want each department manager to control whether any of these sheets are added to their plan file, and how many are added.

Master sheets can be used with templates/plan files, reports, and file group utilities. When using master sheets with templates/plan files, the setup should be performed in the template so that the feature is available to all plan files built with that template. Since workbook protection is always applied to plan files, the master sheets feature is the only way that end users can add sheets to plan files.

How master sheets work

The master sheets feature works as follows:

- Within the file, you create one or more sheets that you intend to designate as master sheets. These sheets should be designed as appropriate for their intended use, including creating calc methods if appropriate (for templates/plan files and file group utilities only). The master sheet can be hidden or visible.
- You designate the master sheet or sheets using the **Master Sheets** setting of the Control Sheet. This setting signals to Axiom that the "add sheet" feature is enabled for this file—or in the case of a template, enabled for the plan files built from the template.
- End users can use the **Add New Sheet** command (in the **File Options** group of the **Axiom** tab) to add a new sheet to the file. The user selects a master sheet to copy and defines a name for the new sheet.

- Axiom creates a copy of the selected master sheet and inserts it into the file. The Control Sheet settings of the master sheet are copied to the new sheet, with a few exceptions:
    - The copied sheet is always flagged as visible, even if the master sheet was hidden.
    - The name of the master sheet is placed in the **Master Sheet** setting for the copied sheet. This associates the copied sheet with the master sheet, and allows the copied sheet to use the same calc method library as the master sheet (if applicable).

  When the sheet is inserted, sheet protection and default views are applied (if configured in the Control Sheet settings). Other "on open" settings are not applied (but will be applied the next time the file is opened).
- When inserting the first copy of the master sheet, the copied sheet is placed after (to the right of) the master sheet if it is visible. If the master sheet is not visible, the copied sheet is placed after the last visible sheet in the workbook.

  Subsequent copies will be inserted after the most recently inserted copy of the master sheet. This means that copied sheets will be displayed in order and continuous within the workbook (as long as these sheets are not manually moved within the file after insertion). Copied sheets must remain visible in the workbook in order to support this behavior.

The user can now edit the copied sheet as needed and as allowed by the sheet design, including using calc methods.

Limitations of master sheets

The following limitations apply to the master sheets feature:

- **Process Plan Files.** It is not possible to run Axiom queries on copied sheets when using Process Plan Files. The list of Axiom queries is generated based on the source templates for the selected plan files, and therefore any queries on copied sheets will not be available for selection because those sheets are only present in the plan files. Selecting the Axiom queries for the master sheet will not cause the corresponding Axiom queries to be run on the copied sheets. The copied sheets will still be calculated during Process Plan Files, and any save-to-database processes on the sheets will be executed (assuming save-to-database is enabled for the process).
- **Print Plan Files.** Print Plan Files is not supported for use with copied sheets. The list of sheets to print is generated based on the source templates for the selected plan files, and therefore any copied sheets will not be available for selection because those sheets are only present in the plan files. The copied sheets can still be printed when the user is in the plan file, using the basic Print functionality.

Design considerations for master sheets

Master sheets are sheet "templates" that end users can copy to create one or more additional sheets in a file. These sheets must be specially designed for use in this environment. Keep in mind the following design considerations when creating sheets that you intend to designate as master sheets.

Control sheet settings and master sheets

When a user adds a new sheet to a file using a master sheet, all Control Sheet settings from the master sheet are copied to the new sheet, with the following exceptions:

- The **Sheet Visible/Hidden** setting for the new sheet is automatically set to **Visible**.
- The name of the master sheet is automatically written to the **Master Sheet** setting for the new sheet.

This copying of Control Sheet settings means that you can define Axiom file features on the master sheet and have those features automatically apply to the copied sheets—such as freeze panes settings, Axiom query settings, and save-to-database settings. If you need any of these settings to be different on the master sheet versus the copied sheets, you can use a formula that looks to the Master Sheet setting. This setting will be blank for the master sheet, but for copied sheets it will contain the name of the master sheet.

When a copied sheet is inserted into the file, sheet protection and default views are applied at that point (if configured on the Control Sheet). Other "on open" settings are not applied (but will be applied the next time the file is opened).

Visibility of the master sheet

In most cases, the visibility of the master sheet should be set to **Hidden**, using the **Sheet Visible/Hidden** setting on the Control Sheet (in the **Sheet Options** section). This assumes that the only purpose of the master sheet is to serve as a "template" for added sheets, and end users do not need to see and work with the master sheet itself.

If the master sheet is hidden, then the master sheet itself will not be visible in the file. However, when a user chooses to add a new sheet using the master sheet, the copied sheet will be automatically flagged as visible.

If you choose to leave the master sheet visible, and if end users are able to make any edits to the master sheet, then those edits will be copied over to any new sheets added using that master sheet. When using master sheets with templates/plan files, keep in mind that copied sheets are made from the master sheet *in the plan file*, not from the master sheet in the source template. Therefore if the master sheet is visible *and editable* in plan files, over time the master sheet may no longer be the same in each plan file.

Calc method libraries and master sheets

For files that support calc method libraries, those libraries can be used with master sheets. All copied sheets created from the master sheet will use the same calc method library as the master sheet. This means that:

- Axiom queries in copied sheets can use the calc method library.
- End users can insert and change calc methods in copied sheets, subject to the normal controls (security permissions and calc method controls).
- The **Apply Calc Method Changes** utility can be used to update calc methods in copied sheets.

This association between copied sheets and the original calc method library is enabled by use of the **Master Sheet** setting on the copied sheets. When a user adds a new sheet using a master sheet, the name of the master sheet is automatically written into this setting. This associates the copied sheet with its source master sheet, and allows the copied sheet to use the same calc method library as the master sheet. Whenever any calc method operations take place on a copied sheet, the master sheet's calc method library is used.

Copied sheets must maintain the same column structure as the original master sheet and its calc method library. End users should not be able to insert or delete columns in the master sheet or any copied sheets when using a calc method library.

Axiom queries and master sheets

When setting up Axiom queries on master sheets, keep in mind the following:

- If Axiom queries on the master sheet are enabled, then those queries will be run when the file is manually refreshed. This may cause the master sheet to be populated with unwanted data that will then be copied to any future sheets created from that master sheet. You may want to dynamically disable the queries on the master sheet but enable them on the copied sheets by using a formula that looks to the **Master Sheet** setting.

  For example, you could use a formula like the following in the **Active** setting for the query:

  ```
  =IF(F35="","Off","On")
  ```

  This would check the Master Sheet cell and turn the query off if it is blank (for the master sheet), but turn it on otherwise (for the copied sheets).

- If an Axiom query is set to refresh on open, that query will not run when a copied sheet is first inserted into the plan file. However, it will be run the next time the file is opened.

- Process Plan Files cannot be used to execute Axiom queries on copied sheets. Because the list of Axiom queries is based on the source templates for the selected plan files, the copied sheets and their Axiom queries are not available. The master sheet and its queries can be available for processing, but selecting queries on the master sheet will not cause the corresponding queries on the copied sheets to be run. You may want to disable **Refresh during document processing** for the Axiom queries on the master sheet so that they are not available for execution in Process Plan Files.

Save-to-database and master sheets

When setting up save-to-database processes on master sheets, keep in mind the following:

- Assuming that the master sheet is being used as a template only, you should set up the save-to-database so that it is only enabled for the copied sheets, not the master sheet. You can do this by using a formula that looks to the Master Sheet setting.

For example, you could use a formula like the following in the **Enabled** setting for the save-to-database process:

```
=IF(F35="","Off","On")
```

This would check the Master Sheet cell and turn the save-to-database off if it is blank (for the master sheet), but turn it on otherwise (for the copied sheets).

- Make sure that the keys on each copied sheet are unique, so that data is not overwritten as the save-to-database for each sheet is processed sequentially. You may need a Detail key in the target table in order to make each sheet's data unique, where the Detail text includes a reference to the current sheet.
- Make sure that the save-to-database process on each copied sheet uses a unique zero tag, to avoid zeroing out data from other save-to-database processes in the file. You may want to use the current sheet name in the zero tag.

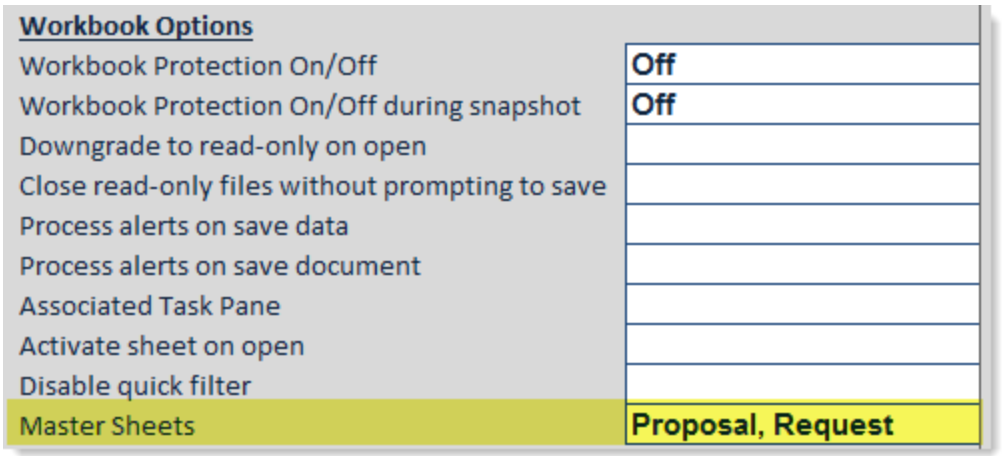
Printing and master sheets

Copied sheets can only be printed when end users are printing within the file using the **Print** button. The **Print Plan Files** option cannot be used to print copied sheets, because the list of sheets to print is built from the source template for the selected plan files. The master sheet will display on the list, but selecting the master sheet will not cause the copied sheets to be printed (and if the master sheet is not visible in the individual plan files, it will not be printed even if selected).

Enabling master sheets for an Axiom file

Once you have created a sheet or sheets that you want to use as master sheets for an Axiom file, you must enable the feature by completing the **Master Sheets** setting on the Control Sheet.

The Master Sheets setting is located in the **Workbook Options** section of the Control Sheet. To designate a sheet in the file as a master sheet, enter the sheet name. If there are multiple master sheets, separate the sheet names with commas.



Workbook Options	
Workbook Protection On/Off	Off
Workbook Protection On/Off during snapshot	Off
Downgrade to read-only on open	
Close read-only files without prompting to save	
Process alerts on save data	
Process alerts on save document	
Associated Task Pane	
Activate sheet on open	
Disable quick filter	
Master Sheets	Proposal, Request

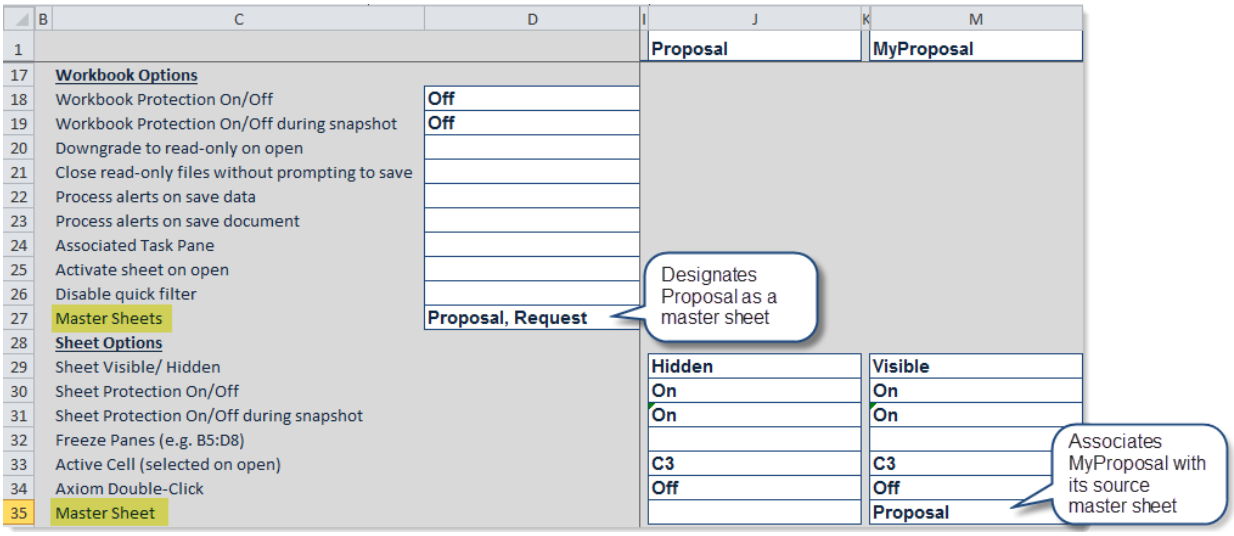
In the example above, the sheets Proposal and Request are designated as master sheets. End users will be able to add new sheets to the file by copying either of these master sheets.

The presence of a valid sheet name in the Master Sheets setting determines whether the **Add New Sheet** command is available to end users. You can use a formula in this setting that changes whether any sheets are shown or which sheets are shown, depending on some criteria. If the Master Sheets setting resolves to blank or to an invalid sheet name, then the Add New Sheet command will not be available.

NOTE: The availability of the Add New Sheet command is determined when the file is opened. If the Master Sheets setting is blank and then you enter a sheet name, you must close and reopen the file before Add New Sheet becomes available.

It is important not to confuse the Master Sheets setting in the Workbook Options with the **Master Sheet** setting in the **Sheet Options**. The Master Sheets setting designates the master sheets in the workbook, whereas the Master Sheet setting is used on the individual copied sheets to associate each copy with its source master sheet. The Master Sheet setting is a system-managed setting that is automatically completed when a sheet copy is added to the workbook; it should not be manually edited.

The following example shows how the Control Sheet of a plan file would look after a sheet is added to the plan file using the Proposal master sheet:



	B	C	D	J	M
1				Proposal	MyProposal
17		Workbook Options			
18		Workbook Protection On/Off	Off		
19		Workbook Protection On/Off during snapshot	Off		
20		Downgrade to read-only on open			
21		Close read-only files without prompting to save			
22		Process alerts on save data			
23		Process alerts on save document			
24		Associated Task Pane			
25		Activate sheet on open			
26		Disable quick filter			
27		Master Sheets	Proposal, Request		
28		Sheet Options			
29		Sheet Visible/ Hidden		Hidden	Visible
30		Sheet Protection On/Off		On	On
31		Sheet Protection On/Off during snapshot		On	On
32		Freeze Panes (e.g. B5:D8)			
33		Active Cell (selected on open)		C3	C3
34		Axiom Double-Click		Off	Off
35		Master Sheet			Proposal

13

Report Wizard

Using the Report Wizard, you can create new reports for your Axiom system. The wizard options guide you through configuring report selections for certain report types.

The Report Wizard can help save time when creating new reports, by defining basic report settings which you can then further modify as desired. However, if you prefer to start with a blank report instead of using the wizard, you can do this by selecting **Reports > New Report**.

Report types

The Report Wizard is available to assist you in creating a variety of reports. You select the basic type of report that you want to create, and then the wizard leads you through some choices to determine the data in the report and the setup of the report. You can also choose to apply standard report options, such as enabling double-click drilling or configuring to refresh on open.

Once the basic report has been created, you can further modify it for your desired formatting, titles, additional calculations, and so on. Depending on the complexity of your desired report and how closely it matches the selected report type, you may only need to make some minor cosmetic modifications, or you may be using the wizard as a starting point from which to make more significant formatting and query modifications.

The Report Wizard supports the following basic types of reports:

- **Free Form**: Display data from any column. No per-record calculations are included. Totals and subtotals apply.

  There are two free-form options: **Dynamic rows** or **Fixed rows**. When using fixed rows, you can specify exactly which dimension elements to include and their order. When using dynamic rows, all records of the specified dimension are dynamically included.

- **Variance**: Compare data from two columns. Per-record variance calculations are included (percent and dollars). Two variance sections are assumed—one for the current period, and one for year-to-date. Totals and subtotals apply.

- **Audit**: Review audit data for a particular table.

Design considerations

When using the Report Wizard, keep in mind the following:

- All Axiom queries built by the wizard are set to rebuild by default. If you want the query to be update-only or update/insert, then you must modify the refresh behavior for the query after the report is created.
- The Control Sheet is hidden by default in reports created by the Report Wizard.

Creating a free-form fixed rows report

Using the Report Wizard, you can create a free-form report with fixed rows. The **Free Form - Fixed Rows** report allows you to:

- Select a single dimension for the rows. You can specify which items in the dimension to include in the rows and their order.

  The row structure of the report is fixed and will not automatically adjust for subsequent additions or deletions to the selected dimension. For example, if the dimension is ACCT.Category and a new category is later added to the ACCT table, this report will not automatically include a row for that category.

- Select any data for the columns. You can also apply a dimension to the column data. For example, you can have a report that contains a column (or columns) for each department or region, for a particular data point.

The Free Form - Fixed Rows report includes totals by row and by column. No other calculations are included in the report.

NOTE: This report uses GetData functions to bring data into the report. Please keep in mind that Axiom queries can also be used to create fixed-row reports, using the "update only" refresh behavior. In many cases the Axiom query construction may be the preferred option, because the data query performance is typically faster.

To create a free-form fixed rows report:

1. On the **Axiom** tab, in the **Reports** group, click **Reports > Report Wizard**.
2. For **Choose a report style**, select **Free Form - Fixed Rows**, and then click **Next**.

3. For **Choose the rows for the report**, select a dimension to define the grouping level ("sum by") for the report, and then click **Next**.

    - For **Dimension**: select the dimensional grouping to use for the rows. By default, the report will have one row for each item in the grouping. You can double-click the box or click the **Choose dimension** icon to select a dimension column. You can select from any reference table.

      Each row in the report will be summed at the selected level. For example, if you select ACCT.ACCT, then each row will be for an individual account. If you select ACCT.Category, then each row will contain the sum for all of the accounts in a category.

    - You can choose whether to **Use all values** of the selected dimension (the default), or to **Manually select values**. If you choose to manually select values, then all values listed in the **Selected Values** box will have a row in the report, in the order listed. You can adjust the order of these items by using the arrow buttons above the box.

4. For **Choose columns to display**, specify the data to display in the columns, and then click **Next**. There are two options to determine the report's columnar structure:

Select one or more columns (default)

Select this option if you want to specify multiple data columns to display in the report, such as four quarterly total columns, or twelve months of budget data.

- In the **Available Columns** list, navigate to the columns that you want to use. You can select from any data table or reference table.
- To select the desired columns, you can double-click the column name or you can click the **Add** button. If a column is added in error, select it and then click **Remove**.

  TIP: If you want to add all columns in a table, select the table name and then click **Add**.

- In the resulting report, the columns will be displayed in the order listed in the **Selected Columns** box. You can adjust the order by using the arrow buttons above the box.

Repeat data with dimension filters

Select this option if you want the columns to display data by a particular dimensional grouping—for example, show YTD data by department, region, or VP. You can use this to compare data for all items in a dimension or dimensional grouping.

- For **Data**, select a single data column to define the data for the report. You can double-click the box or click the **Choose column** icon to select a column. You can select from any data table that uses the selected row dimension.

NOTE: The Report Wizard limits the data selection to one column for simplicity. It is possible to manually create reports that use any number of data columns per dimensional grouping.

- For **Dimension**: select the dimensional grouping to use for the columns. The report will have one column for each item in the grouping. You can double-click the box or click the **Choose dimension** icon to select a dimension column. You can select from any reference table that the data table links to.
- Specify whether to **Use all values** of the selected dimension grouping, or to **Manually select values**. If you choose to manually select values, then all values listed in the **Selected Values** box will have a column in the report, in the order listed. You can adjust the order of these items by using the arrow button above the box.

For example, if you choose the alias CYA_YTD for the data, and DEPT.Region for the dimension, then the report will have one column for each region, showing the CYA_YTD data for each region.

NOTE: The column structure created by the Report Wizard is not dynamic—meaning, the dimension elements will be "hard-coded" into the columns and will not automatically update if dimension elements are added or deleted in the future. It is possible to manually create a report with a dynamic columnar structure using a horizontal Axiom query. For more information, see Using horizontal Axiom queries.

If you do not want to specify any other report options, you can click **Finish** at this point to create the report.

5. Optional. For **Specify a filter**, define a filter to limit the data in the report, and then click **Next**. You can type the filter or use the Filter Wizard.

   The filter will be added as a sheet filter to the report. You can always edit the report later to apply a filter if you do not want to apply one now.

6. For **Choose report options**, specify any report options that you want to apply to the report, and then click **Finish**.

   These are all standard report options. If you do not select them now, you can enable them later using the Sheet Assistant or the Control Sheet.

The report is created according to your selections. You can now modify it as desired.

NOTE: Depending on the total number of GetData functions in the generated report, it may take some time to initially open and calculate.

Free-form fixed rows report example

The following is an example of a report generated using the free-form fixed rows option.

2012 Budget by Region					
Region	Jan-12	Feb-12	Mar-12	Apr-12	May-12
China	7,232,892	6,805,226	6,299,662	5,785,335	5,596,460
Corporate	2,605,078	2,716,598	2,786,138	2,738,498	2,687,638
France	-	-	-	-	-
India	84,510	82,150	82,410	81,750	82,910
Italy	63,520	46,851	19,296	19,194	19,335
Singapore	3,793,166	3,109,735	4,931,512	4,174,990	3,570,425
UK	337,910	201,944	160,322	401,184	220,605
US Central	12,223,625	11,433,858	16,144,580	14,335,016	16,485,625
US East	40,205,625	28,512,180	29,057,512	27,966,571	29,762,970
US West	4,333,320	5,081,244	2,459,814	2,078,878	1,077,410
Total	70,879,646	57,989,786	61,941,246	57,581,417	59,503,378

Creating a free-form dynamic rows report

Using the Report Wizard, you can create a free-form report with dynamic rows. The **Free Form - Dynamic Rows** report allows you to:

- Select up to two dimensions to dynamically populate the rows. If two dimensions are selected, the first dimension is grouped by the second dimension—for example, accounts subtotaled by account categories.

  The row structure of this report is dynamic and will automatically adjust for subsequent additions or deletions to the selected dimension. For example, if the dimension is ACCT.ACCT and new accounts are later added to the ACCT table, this report will automatically update to include new rows for those accounts. (Assuming that the new accounts meet any filter applied to the report, and that the query is still configured to rebuild or insert data.)

- Select any data for the columns. You can also apply a dimension to the column data. For example, you can have a report that contains a column (or columns) for each department or region, for a particular data point.

The Free Form - Dynamic Rows report does not contain any row-based calculations. Full-column totals or section subtotals are included.

To create a free-form dynamic rows report:

1. On the **Axiom** tab, in the **Reports** group, click **Reports > Report Wizard**.
2. For **Choose a report style**, select **Free Form - Dynamic Rows**, and then click **Next**.
3. For **Choose the rows for the report**, select a grouping dimension to define the rows and optionally a subtotal dimension, and then click **Next**.
   - For **Choose a dimension or grouping column to define the rows**, select the column to define the sum level of the rows. You can select from any reference table in the system. You can double-click inside the box to bring up the column chooser, or you can click the **Choose dimension** icon to the right of the box.

For example, if you choose ACCT.ACCT, then each individual row will contain the data for a single account. If you choose ACCT.Category, then each individual row will contain the data summed by account categories.

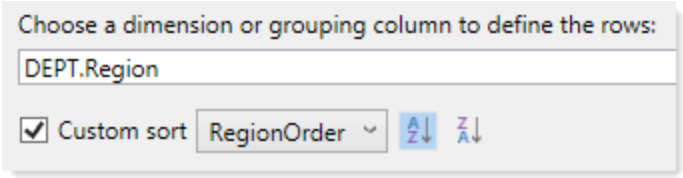
- Optionally, you can select a second dimension by which to group and subtotal the rows of the first dimension. To do this, select the **Use subtotal sections** check box. Then for **Choose a dimension or grouping column to define the sections**, select the column that you want to use. The same rules apply as when selecting the first dimension.

  For example, if you choose ACCT.ACCT as the grouping dimension and then ACCT.Category as the subtotal dimension, then the report will display with accounts listed under their respective categories. A subtotal is included for each category section.

The following additional options are available for the row dimensions:

Custom sort

By default, the selected dimensions are sorted in ascending order. If desired, you can specify a **Custom sort** to change the order to descending, or even to specify a different grouping column to use for the sort.



If this check box is selected for a dimension, then the following new options display to the right of the check box:

- **Sort column:** Select the column to sort by. By default, this is the same column chosen as the dimension. You can change this to a different column if desired.

  For example, you may want to display regions in your report, but those regions should display in a different order than alphabetical by region. You could have another column in the table such as RegionOrder which defines the sort order for the regions.

- **Ascending / descending:** You can click the **Sort ascending** or **Sort descending** buttons to the right of the column selection to determine the sort order.

Insert new sections on refresh

The option **Insert new sections on refresh** determines how the subtotal groupings will be created. This option is only available if you have selected a subtotal dimension.

- If selected, then the subtotal groupings are created by using nested Axiom queries. The first query dynamically builds out a second query with a data range for each subtotal section. Therefore if new items are added to the subtotal dimension (or old items removed), the query will adjust automatically when refreshed.
- Otherwise, the subtotal groupings are created using a single Axiom query with a data range for each section. These data ranges are static and will not adjust for future additions or deletions to the subtotal dimension.

For example, imagine you have a report with accounts subtotaled by account categories. A few months later, a new account category is added to your system. If **Insert new sections on refresh** was selected, then when the report is refreshed, the nested query is dynamically rebuilt and a new section (data range) is automatically added for the new category.

If **Insert new sections on refresh** was *not* selected, then the query sections are static and the new category is not included when the report is refreshed. The report will not contain data for any accounts that were assigned to that new category. You would need to manually adjust the report to add a data range for the new category.

> **NOTE:** This assumes that the refresh settings and basic query structure of the original report were left as is after the report was created. The **Insert new sections on refresh** option only determines how the report is originally structured—once created, the report can be adjusted as needed.

4. For **Choose columns to display**, specify the data to display in the columns, and then click **Next**. There are two options to determine the report's columnar structure:

Select one or more columns (default)

Select this option if you want to specify multiple data columns to display in the report, such as four quarterly total columns, or twelve months of budget data.

- In the **Available Columns** list, navigate to the columns that you want to use. You can select from any data table or reference table.
- To select the desired columns, you can double-click the column name or you can click the **Add** button. If a column is added in error, select it and then click **Remove**.

  > **TIP:** If you want to add all columns in a table, select the table name and then click **Add**.

- In the resulting report, the columns will be displayed in the order listed in the **Selected Columns** box. You can adjust the order by using the arrow buttons above the box.

Repeat data with dimension filters

Select this option if you want the columns to display data by a particular dimensional grouping—for example, show YTD data by department, region, or VP. You can use this to compare data for all items in a dimension or dimensional grouping.

- For **Data**, select a single data column to define the data for the report. You can double-click the box or click the **Choose column** icon to select a column. You can select from any data table that uses the selected row dimension.

  NOTE: The Report Wizard limits the data selection to one column for simplicity. It is possible to manually create reports that use any number of data columns per dimensional grouping.

- For **Dimension**: select the dimensional grouping to use for the columns. The report will have one column for each item in the grouping. You can double-click the box or click the **Choose dimension** icon to select a dimension column. You can select from any reference table that the data table links to.
- Specify whether to **Use all values** of the selected dimension grouping, or to **Manually select values**. If you choose to manually select values, then all values listed in the **Selected Values** box will have a column in the report, in the order listed. You can adjust the order of these items by using the arrow button above the box.

For example, if you choose the alias CYA_YTD for the data, and DEPT.Region for the dimension, then the report will have one column for each region, showing the CYA_YTD data for each region.

NOTE: The column structure created by the Report Wizard is not dynamic—meaning, the dimension elements will be "hard-coded" into the columns and will not automatically update if dimension elements are added or deleted in the future. It is possible to manually create a report with a dynamic columnar structure using a horizontal Axiom query. For more information, see Using horizontal Axiom queries.

If you do not want to specify any other report options, you can click **Finish** at this point to create the report.

5. Optional. For **Specify a filter**, define a filter to limit the data in the report, and then click **Next**. You can type the filter or use the Filter Wizard.

   The filter will be added as a sheet filter to the report. You can always edit the report later to apply a filter if you do not want to apply one now.

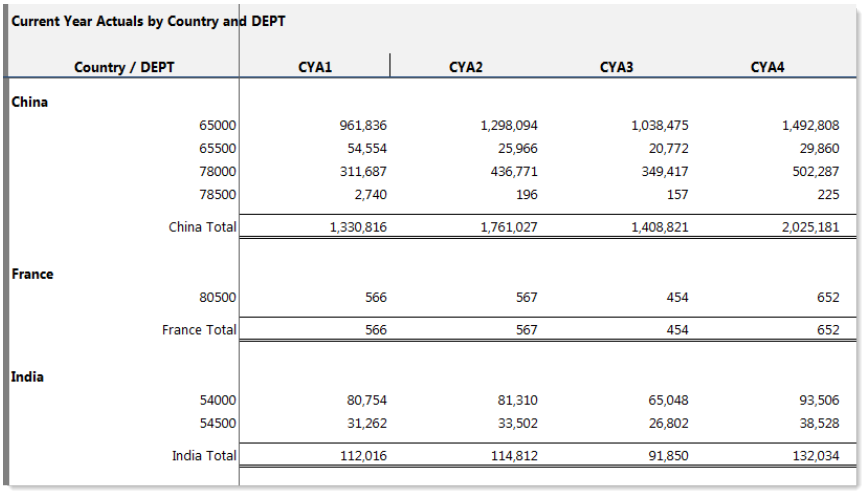
6. For **Choose report options**, specify any report options that you want to apply to the report, and then click **Finish**.

   These are all standard report options. If you do not select them now, you can enable them later using the Sheet Assistant or the Control Sheet.

The report is created according to your selections. You can now modify it as desired.

Free-form dynamic rows report example

This example uses the option to define two row dimensions for subtotal groupings. The "base" dimension is DEPT and the subtotal dimension is DEPT.Country.



Current Year Actuals by Country and DEPT

Country / DEPT	CYA1	CYA2	CYA3	CYA4
China				
65000	961,836	1,298,094	1,038,475	1,492,808
65500	54,554	25,966	20,772	29,860
78000	311,687	436,771	349,417	502,287
78500	2,740	196	157	225
China Total	1,330,816	1,761,027	1,408,821	2,025,181
France				
80500	566	567	454	652
France Total	566	567	454	652
India				
54000	80,754	81,310	65,048	93,506
54500	31,262	33,502	26,802	38,528
India Total	112,016	114,812	91,850	132,034

Creating a variance report

Using the Report Wizard, you can create a variance report. The defining characteristic of the variance report is the variance calculations that are automatically included in the report.

The variance report allows you to:

- Select up to two dimensions for the rows. If two dimensions are selected, the first dimension is grouped by the second dimension—for example, accounts grouped by account categories.
- Select columns for two sections of variance calculations. Each section can compare any two columns, but by default the report assumes a budget-to-actuals comparison, with the first section comparing the current period, and the second section comparing year-to-date (YTD). Both sections include variance calculations by dollar amount and percent, for each row.

Full-column totals or section subtotals are included.

To create a variance report:

1. On the **Axiom** tab, in the **Reports** group, click **Reports > Report Wizard**.
2. For **Choose a report style**, select **Variance**, and then click **Next**.
3. For **Choose the rows for the report**, select up to two dimensions to define the grouping level ("sum by") for the report, and then click **Next**.
    - For **Choose a dimension or grouping column to define the rows**, select the column to define the sum level of the rows. You can select from any reference table in the system. You

can double-click inside the box to bring up the column chooser, or you can click the **Choose dimension** icon to the right of the box.

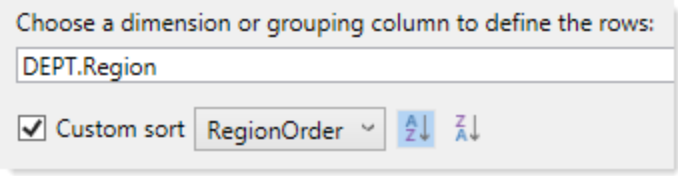
For example, if you choose ACCT.ACCT, then each individual row will contain the data for a single account. If you choose ACCT.Category, then each individual row will contain the data summed by account categories.

- Optionally, you can select a second dimension by which to group and subtotal the rows of the first dimension. To do this, select the **Use subtotal sections** check box. Then for **Choose a dimension or grouping column to define the sections**, select the column that you want to use. The same rules apply as when selecting the first dimension.

  For example, if you choose ACCT.ACCT as the grouping dimension and then ACCT.Category as the subtotal dimension, then the report will display with accounts listed under their respective categories. A subtotal is included for each category section.

Custom sort

By default, the selected dimensions are sorted in ascending order. If desired, you can specify a **Custom sort** to change the order to descending, or even to specify a different grouping column to use for the sort.



If this check box is selected for a dimension, then the following new options display to the right of the check box:

- **Sort column:** Select the column to sort by. By default, this is the same column chosen as the dimension. You can change this to a different column if desired.

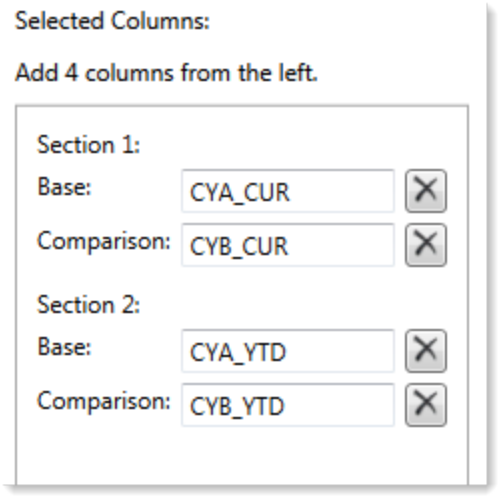
  For example, you may want to display regions in your report, but those regions should display in a different order than alphabetical by region. You could have another column in the table such as RegionOrder which defines the sort order for the regions.

- **Ascending / descending:** You can click the **Sort ascending** or **Sort descending** buttons to the right of the column selection to determine the sort order.

4. For **Choose variance columns**, select the four columns that you want to compare, and then click **Next**.

   The variance report uses two sections of columns. Each section has a base column and a comparison column—for example, the base can be an actuals column and the comparison can be a budget column.

By default these sections will be automatically completed as follows, assuming that your system has alias columns that follow these conventions (with or without the underscore).



If you want to compare different columns, or if your system does not use these conventions, then select the desired columns as follows:

- If necessary, click the **Delete** button to the right of the box to delete any existing value.
- In the **Available Columns** list, navigate to the columns that you want to use. The default view is alias names, but you can change the view to search by tables, table types, or table folders. You can select from any data table. You can use the filter box above the list to find a specific table and column.
- To select the desired columns, you can double-click the column name or you can click the **Add** button. Selections are placed in empty column boxes in the order they are listed.

  You can also double-click in any column box to bring up the **Column Chooser** dialog, to select or change a particular column.

All four columns are required. You can modify the report after it is created if needed.

NOTE: If you do not want to specify any other report options, you can click **Finish** at this point to create the report.

5. Optional. For **Specify a filter**, define a filter to limit the data in the report. You can type the filter or use the Filter Wizard.

   The filter will be added as a sheet filter to the report. You can always edit the report later to apply a filter if you do not want to apply one now.

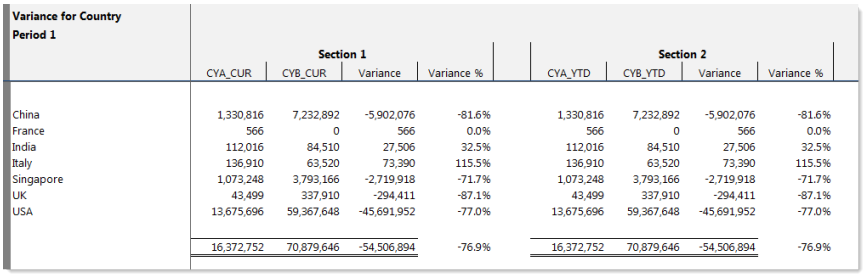
6. For **Choose report options**, specify any report options that you want to apply to the report, and then click **Finish**.

   These are all standard report options. If you do not select them now, you can enable them later using the Sheet Assistant or the Control Sheet.

The variance report is created according to your selections. You can now modify it as desired.

Variance report example

The following is an example of a report generated using the variance option.



Variance for Country Period 1	Section 1				Section 2			
	CYA_CUR	CYB_CUR	Variance	Variance %	CYA_YTD	CYB_YTD	Variance	Variance %
China	1,330,816	7,232,892	-5,902,076	-81.6%	1,330,816	7,232,892	-5,902,076	-81.6%
France	566	0	566	0.0%	566	0	566	0.0%
India	112,016	84,510	27,506	32.5%	112,016	84,510	27,506	32.5%
Italy	136,910	63,520	73,390	115.5%	136,910	63,520	73,390	115.5%
Singapore	1,073,248	3,793,166	-2,719,918	-71.7%	1,073,248	3,793,166	-2,719,918	-71.7%
UK	43,499	337,910	-294,411	-87.1%	43,499	337,910	-294,411	-87.1%
USA	13,675,696	59,367,648	-45,691,952	-77.0%	13,675,696	59,367,648	-45,691,952	-77.0%
	16,372,752	70,879,646	-54,506,894	-76.9%	16,372,752	70,879,646	-54,506,894	-76.9%

Creating an audit report

Using the Report Wizard, you can create a report that shows the available audit information for the records in a particular table. If a record has been changed, and that change is within your system's configured audit history, this report will display the changed data, the user who changed it, and when it was changed.

NOTE: The audit tables track changes to the *data* in the table, not changes to table properties. For information on changes to table properties, use the Axiom Audit Manager.

To create an audit report:

1. On the **Axiom** tab, in the **Reports** group, click **Reports > Report Wizard**.
2. For **Choose a report style**, select **Audit**, and then click **Next**.
3. For **Choose a table**, complete the following settings, and then click **Next**.
    - For **Table**, specify the table for which you want to view audit data. You can type the table name, or you can click the **Choose Table** button to select a table. The table must be enabled for auditing in order to report on audit data.
    - Optionally, you can filter the report by date. By default, the report will contain **All Audit History**, which means that all *available* audit history for the table will be included in the report. If desired, you can select **Date Range** and then specify a **Start** and/or **End** date for the report.

      The date range only applies to historical records. It will be applied as a filter to the Axiom query against the audit table. The most current record for the table is always displayed in the report.

IMPORTANT: The historical data available to you depends on how long audit data is configured to be retained in your system. It is possible to select dates for which audit data does not exist—this does not mean that no changes were made in that time period, it means that the audit data for that time period has been deleted from the system. If you have questions about the available audit data in your system, please contact your system administrator.

NOTE: If you do not want to specify any other report options, you can click **Finish** at this point.

4. Optional. For **Specify a filter**, define a filter to limit the data in the report. You can type the filter or use the Filter Wizard .

   The filter will be added as a data filter to the Axiom query against the main table. You can always edit the report later to apply a filter if you do not want to apply one now.

5. Optional. For **Choose report options**, specify any report options that you want to apply to the report, and then click **Finish**.

   These are all standard report options. If you do not select them now, you can enable them later using the Sheet Assistant or the Control Sheet.

Using the audit report

The audit report displays all current records in the selected table. If audit history exists for a particular record, it is grouped underneath the current record. You can expand the section to view the history.

You can tell whether a record has audit history by checking the Previous Versions column of the report. If this column is 0, then no audit history exists for the record. Otherwise, this column displays the number of previous versions of that record stored in the audit history.

Audit History for DEPT

DEPT	Description	Template	WorldRegion	Country	Region	Currency	VP	Modified By	Modified Date	Previous Versions
10000	Strat Plan	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	12/4/2012 14:23	0
11000	Balance Sheet	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0
20000	Corporate	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0
21000	Corporate Administration	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0
22000	Information Technologies	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/19/2012 11:03	0
23000	Purchasing & Materials Mgmt	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	2/19/2013 10:21	1
	Purchasing & Materials Mgmt	Master Budget Template	Corporate	USA	Corporate	USD	Rick Simpson	whunter	7/10/2012 13:09	
24000	Business Development	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:09	0
25000	Finance	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:09	0
26000	Portfolio Management	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0
27000	Human Resources	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0
28000	Facilities	Master Budget Template	Corporate	USA	Corporate	USD	Mike Choi	whunter	7/10/2012 13:10	0

Example audit report

NOTE: Deleted records are not shown in this report. If you want to view the audit history of a deleted record, you can build an Axiom query that directly queries the audit table.

The audit report includes all columns in the specified table, including the ModifiedBy and ModifiedDTM columns.

14

File Setup Tools

Several tools are available in Axiom files to assist you with setup tasks for Axiom files. These tools may be helpful when:

- Building Axiom queries
- Building data sources
- Completing Control Sheet settings
- Inserting GetData functions
- Setting up a save-to-database
- Troubleshooting file setup

Using the Sheet Assistant

The Sheet Assistant contains tools to assist you in configuring Axiom queries and Control Sheet settings for a sheet. For more information on the specific settings, see Control Sheet Settings and Axiom query settings.

NOTE: In order to see and use the Sheet Assistant for a file, you must be an administrator or have the **Allow Sheet Assistant** security permission for the file. The Sheet Assistant is not available for non-managed files.

Working in the Sheet Assistant

The Sheet Assistant contains three sections. You can expand or collapse a section of the pane by clicking on the arrow icon next to the section name.

- **Worksheet Settings**: View and edit certain Control Sheet settings for the current sheet.
- **Axiom Queries**: View and edit certain Axiom query settings for the current sheet.
- **Columns / Aliases**: Look up table and column names, and drag and drop them into the current sheet (for example, to build the field definition).

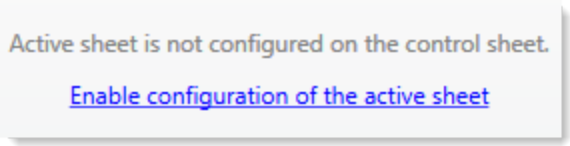
For the worksheet settings and the Axiom query settings, the pane displays the settings as currently defined in the Control Sheet. As you edit the settings in the pane, the Control Sheet is updated for your changes, and vice versa. You must save the file in order to save any changes made within the current session.

You can use cell references and formulas just as you can when defining settings in the Control Sheet. The pane displays formula results by default, but if you click in a field that contains a formula, you can edit the formula.

You can double-click any field name to be taken directly to that setting in the Control Sheet. For example, you can double click the label **Sheet Protection** to be taken to that setting in the Control Sheet for the current sheet.

Adding a sheet to the Control Sheet

If you are on a sheet that is not currently set up on the Control Sheet, then the following message displays at the top of the Sheet Assistant:



You can click **Enable configuration of the active sheet** to automatically add the sheet to the Control Sheet and begin editing settings.

Updating the Control Sheet

If the current Control Sheet is an older version, and the file is currently open with read/write access, then the following message displays at the top of the Sheet Assistant:

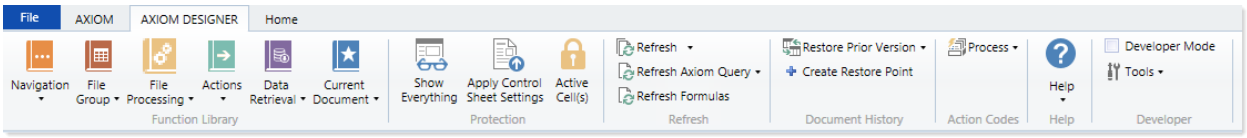
You can click **Update the control sheet** to automatically update the current Control Sheet to the latest version. For more information, see Updating a Control Sheet.

You are not required to update the Control Sheet. The file will remain operational as is, and you can continue to use the Sheet Assistant to modify settings that do exist on the current Control Sheet.

If a setting does not exist on the current Control Sheet, then the Sheet Assistant displays the default value for that setting. If you attempt to modify a setting that does not exist on the Control Sheet, you will be prompted to perform the update.

Using the Axiom Designer tab

Use the features on the Axiom Designer tab to assist with setting up Axiom files. The tab contains various tools and special commands intended to be used for file development and testing.



By default, the Axiom Designer tab is only available to administrators, however, your organization can choose to expose the tab to other users if desired. Additionally, the contents of the tab can also be customized.

The following sections detail the default contents of the Axiom Designer tab.

Function Library

Use the Function Library to insert an Axiom function into the currently selected cell. The library is organized into the following sections:

Item	Description
Navigation	Functions used to link to other files, such as: GetDocumentHyperlink, GetDocument, and all functions relating to creating hyperlinks in Axiom forms
File Group	Functions relating to file groups, such as: GetFileGroupID, GetFileGroupVariable, GetCalcMethod, and GetPlanFileAttachment
File Processing	Functions used in file processing, such as IsRunningMultipass and GetCurrentValue
Actions	Functions that perform an action, such as RunEvent
Data Retrieval	Functions that retrieve data from the database or system data, such as GetData, GetDataElement, GetPeriod, GetSecurityInfo, GetSystemInfo, and GetUserInfo
Current Document	Functions that return information about the current document, such as GetColumnLetter, GetRowNumber, and GetDocumentInfo

Clicking on a function in the library places a "starter" version of the function into the current cell, and opens the function wizard. For most functions this is the Excel function wizard (available in the Excel Client only); however some functions have a specialized wizard available, such as GetData and GetDocumentHyperlink. You can complete the function parameters using the wizard, or you can click **OK** to close the wizard and complete the function parameters manually.

Protection

Use the features in the Protection group to turn off or apply various protections and display settings in the file.

Item	Description
Show Everything	Turns off all protections and display controls in the file, whether applied via Axiom functionality or manually. Workbook and worksheet protections are disabled, frozen panes are removed, all sheets are unhidden, and all rows and columns are unhidden. This is intended to be used as a one-click solution to display everything in the file so that you can perform further file development. Any protections and controls that were configured via Axiom functionality remain configured, and will be reapplied the next time the file is opened (or by clicking **Apply Control Sheet Settings**). Any protections and controls that were applied manually are now removed. This feature is only available to administrators or to users with unprotect permissions to the file.
Apply Control Sheet Settings	Turns on all configured protections and display controls in the file, including workbook/worksheet protection, sheet visibility, freeze panes, and the default view. If you had protections or controls in the file that were manually applied (instead of by using the Control Sheet), and then you removed them using **Show Everything**, then using **Apply Control Sheet Settings** will not reapply those manual settings. This feature is only available to administrators or to users with unprotect permissions to the file.
Active Cell(s)	Indicates whether the currently selected cell or cells are locked. • A locked lock icon indicates that the cells are locked. • An unlocked lock icon indicates that the cells are unlocked. • A grayed out lock icon indicates that some cells are locked and some are unlocked. You can change the configuration of the selected cells to be locked or unlocked by clicking the button. Generally speaking, a cell must be unlocked if you want to allow a user to edit the cell (assuming sheet protection is applied to the sheet). Certain Axiom form features also require target cells to be unlocked, in order to update the sheet with the current component status. This toggle is provided to make it easier for you to see and change the locked status of a cell.

Refresh

Use the features in the Refresh group to perform advanced refresh actions.

Item	Description
Refresh	This is the same as the Refresh button on the main Axiom tab, replicated here for convenience. You can refresh the entire workbook or just the active sheet.
Refresh Axiom Query	Refresh a single selected Axiom query. The drop-down list displays all active Axiom queries defined in the file, organized by sheet. Only the selected Axiom query will be run; formulas are still refreshed as normal.
Refresh Formulas	Refresh only the formulas in the file. No Axiom queries will be run.

Document History

Use the features in the Document History group to create document versions and to restore prior document versions.

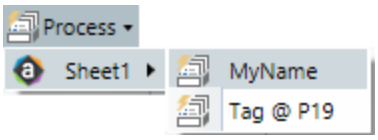
Item	Description
Restore Previous Version	Restores a selected document version. You can select from any available document version. **IMPORTANT:** Using this feature will *replace* the current document with the selected version, it will not open the version in a separate tab. If the current document has unsaved changes, you will first be prompted to save. Once the selected version is opened, if you want to save it as the current version you must save the document. If you close without saving, then the document will remain at the state it was in before you opened the prior version.
Create Restore Point	Saves the document as a new document version, so that if needed you can restore it later using **Restore Previous Version**. Normally, a new document version is created only once per lockout session. The **Save and Create Restore Point** feature allows you to create as many versions as needed within the current session, so that you can later return to any point of the file.

NOTE: These features are not related to the separate "restore point" feature for plan files. For more information on that feature, see the *File Group Administration Guide*.

Action Codes

Use the Action Codes group to process action codes on demand (independent of running Axiom queries or other means of triggering action codes).

Clicking the **Process** button brings up a list of ActionCodes tags in the document, organized by sheet. If the tag is named, it displays by name, otherwise it displays based on location in the sheet. The following example shows one named tag (MyName) and one unnamed tag (located in cell P19):



Clicking on a tag name or location will process the actions just for that particular ActionCodes tag. This is intended to assist in action code setup and troubleshooting.

Developer

Use the features in the Developer group to assist in file development.

Item	Description
Developer Mode	Enable or disable Developer Mode for the current session. Developer Mode provides access to certain features and tools to help test and troubleshoot your files. For more information, see Using Developer Mode.
Tools	The Tools menu provides access to the following features: • **Add a Control Sheet**: Add any type of control sheet to the current file. • **Find Formula Errors**: Search the current file for formula errors. • **Refresh File System**: Refresh the current file system. Sometimes this is necessary to update the file list on the Reports menu or other items. • **Clear UDF Data Cache**: Clear the UDF cache. This may be necessary when testing or troubleshooting formula-related issues, to start with a clean cache. • **Process Alerts**: Process alerts defined in the current file. The Tools menu is only visible to administrators, with one exception: users with access to Process Alerts on the Axiom tab will also see it here if they have access to the Axiom Designer tab.
Clean Document	Clears the following items in the document: • Time stamps in the **Last refresh time** field for Axiom queries (used when **Refresh only if primary table changed since last refresh** is enabled) • The Result and IsError columns for any DataLookup data sources

Using the Data Source Assistant

The Data Source Assistant can help you build and edit data sources in Axiom. Currently, the Data Source Assistant works with the following data source types:

- DataLookup
- Grid
- KPISource
- RefreshVariables

Using the Data Source Assistant, you can add tags to a data source, and add / edit content in a data source. For example, you can add new variables to a RefreshVariables data source or edit existing variables.

The Data Source Assistant is available to administrators and to any user with the **Sheet Assistant** permission to a file. It is visible whenever the Sheet Assistant is also visible.

The Data Source Assistant is divided into three sections:

- **Data Source Picker**: Lists the current data sources in the file so that you can navigate to them.
- **Edit Tags**: Allows adding tags to the current data source.
- **Selection Editor**: Allows editing the current selection. This feature varies depending on the type of data source.

Data Source Picker

This section lists the data sources that currently exist in the file. Only supported data sources are listed. All unsupported data sources are ignored, because you cannot currently work with these data sources using the Data Source Assistant.

To go to a data source, double-click the row in the grid. You will be taken to that data source, and your cursor will be placed in the primary tag.

Data sources are listed by name, type, and sheet address:

- Grid and KPISource data sources are required to be named, so each one can be differentiated by name as well as by sheet address.
- RefreshVariables data sources are not named, but each file can only have one of these data sources.
- DataLookup data sources are not required to be named. Unnamed DataLookup data sources can only be differentiated by sheet addresses.

Currently it is not possible to create new data sources using the Data Source Assistant. You must use other helper tools to create them, or create them manually. However, once the primary tag has been added to a sheet, you can then insert all remaining column and row tags using the Data Source Assistant.

Edit Tags

You can use this section to edit the row and column tags for the current data source. Each type of data source supports different row and column tags, however, they all can be edited as follows:

Action	Description
Insert	Inserts a new column or row at the current cursor location and adds the tag. The tag is added within the control row or control column as appropriate. This action is available when the data source does not already contain a tag of this type, or the data source can contain multiple tags of this type.
Update	Adds the tag in the current column or row, overwriting any current contents. The tag is added within the control row or control column as appropriate. This action is available when the data source does not already contain a tag of this type, or the data source can contain multiple tags of this type.

Selection Editor

You can use this section to edit the current selection in the data source. This section varies depending on the data source type.

Data Source Type	Description
DataLookup	Place your cursor anywhere in a data lookup row to edit the properties of that row. Your edits are written back to the appropriate cells in the data source.
Grid	Formatted grid contents can be edited as follows: • Place your cursor in a `[ColumnStyle]` row or in the `[RowStyle]` column to view available styles and add styles to the data source. • Place your cursor in a cell within a tagged `[Row]` and `[Column]`—including `[Fixed]` rows and columns—to insert or edit a content tag.
KPISource	Place your cursor anywhere in a KPI row to edit the properties of that row. Your edits are written back to the appropriate cells in the data source.
RefreshVariables	Place your cursor anywhere in a variable row to edit the properties of that row. Your edits are written back to the appropriate cells in the data source.

Using the Filter Wizard

The Filter Wizard is available throughout the system to assist you in constructing filters. The Filter Wizard offers two different approaches for building filters:

- **Hierarchies**: Build a filter using hierarchies that have been set up for your system. You select the

items that you want to include and the Filter Wizard builds the filter criteria statement for you.

- **Advanced Filter**: Build a filter using any table and column that is relevant to the current context. This approach also allows for more operators, including greater than, less than, and not equal to.

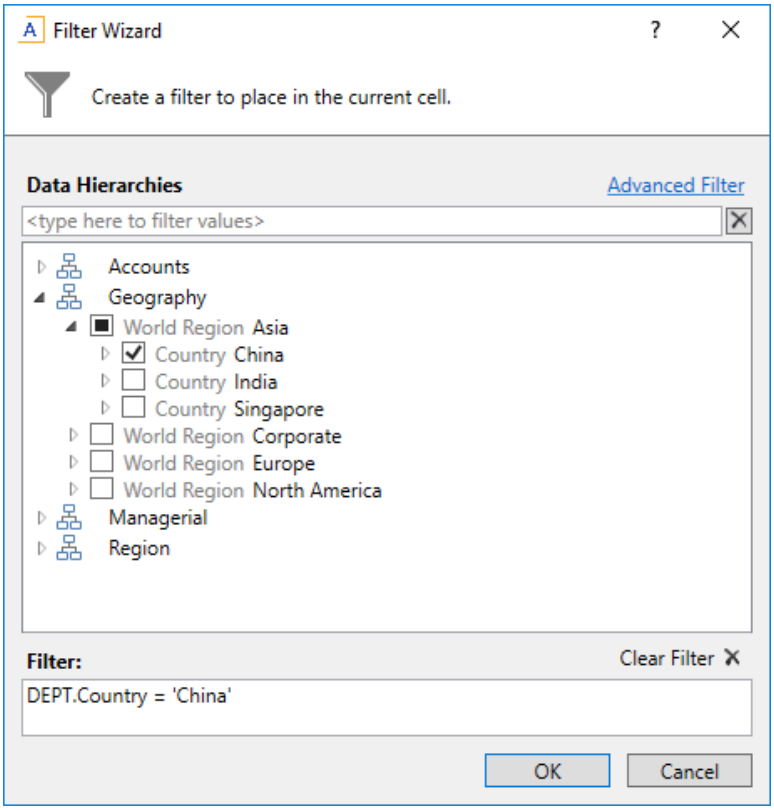
  The Advanced Filter dialog has two versions, depending on how the dialog is launched. The "desktop" version of the dialog is used in most places in the Desktop Client. The "web" version is used in the Web Client and with certain special features in the Desktop Client.

The Filter Wizard can be launched by using the Filter button in various dialogs and task panes. You can also open the wizard manually from any cell in an Axiom spreadsheet file, using the right-click menu: **Axiom Wizards > Filter Wizard**.

Hierarchies

To create a filter using hierarchies, select the check box for each item that you want to include in the filter. You can expand each hierarchy to see the items listed in it. You can also type a value into the filter box above the hierarchies to filter the list.

For example, you may have a hierarchy for Geography that starts at the WorldRegion level, then goes down to the Country level, and then goes down to the LocalRegion level. If you want to filter by a particular country in the Asia WorldRegion, you can expand the Geography hierarchy, then expand the Asia WorldRegion, and then select the desired country.



As you select items, the filter criteria statement is created in the **Filter** box at the bottom of the dialog. You can click **OK** to apply the filter as is, or you can manually edit the filter by typing in the box.

The hierarchies available to you are defined by your system administrator, based on grouping columns in reference tables. If your system has no defined hierarchies (or if no defined hierarchies are relevant to the current context), then the **Data Hierarchies** section does not display, and the Advanced Filter opens directly.

Note the following about filters created using data hierarchies:

- Only "include" filter criteria statements can be created using data hierarchy selections. As you select items, those items will be included by using an equals (=) operator or an IN statement (for including multiple items at the same level). If you want to write a filter criteria statement that specifies items to exclude, or that uses other operators such as greater than or less than, then you must use the Advanced Filter.
- Certain assumptions are made regarding the use of AND and OR when multiple items are selected from different hierarchy levels or different hierarchies. If you want to change the way each statement is joined, you can manually edit the filter in the Filter box, or you can use the Advanced Filter.
- Sometimes when you select a "child" item underneath a "parent" item, the child and parent will be joined with AND. For example: `DEPT.VP='Jones' AND DEPT.Manager='Smith'`. This means that the DEPT table has other instances of Manager Smith that belong to different VPs, so the compound statement is to ensure that you only get the data where Manager Smith is under VP Jones. (You can manually edit the filter to remove the Jones portion of the statement if you want to see all data for Manager Smith, regardless of VP). If instead Axiom constructs the filter as just `Dept.Manager='Smith'`, that means all instances of Manager Smith are also under VP Jones.

If an existing filter was present when the dialog was launched, this filter is listed in the Filter box. You can manually edit this filter if desired, or you can make new selections to overwrite the current filter.

Advanced filter (desktop version)

Using the desktop version of the Advanced Filter dialog, you can create a filter using any relevant table and column, and using any supported operator. The desktop version is used in most areas of the Desktop Client.

When using the desktop version, you can see a link in the top right corner named **Simple Filter**. Clicking that link switches the dialog view to Hierarchy view. Once you are in Hierarchy view, you can switch back to Advanced Filter view by clicking the **Advanced Filter** link in the top right corner. Only the desktop version of the dialog supports switching between Advanced Filter and Hierarchies.

To create a filter using the desktop Advanced Filter dialog:

1. In the left-hand side of the dialog, select the table column on which you want to base the filter.

For example, if you want to create a filter such as `DEPT.DEPT>=5000`, then you must select the DEPT column from the DEPT table.

To find the desired table and column, you can do the following:

- Use the **View by** option to view the list by table, table type, folder, or alias. Note that if you want to select an alias, you must change the view to Alias—aliases are not listed under their assigned table.
- You can also filter the list by typing into the filter box. The filter matches based on table name or column name.

Once you select a table column, the values in that column display in the right-hand side of the dialog.

TIP: Alternatively, you can use the folder icon to the right of the **Preview** box to load a previously saved filter from the Filters Library. If you do this, your selected filter is placed in the Preview box, overwriting any current content in the preview. Skip to step 4.

2. In the right-hand side of the dialog, type or select the value on which you want to base the filter.

   You can type into the box above the list of values to filter the list, or to specify a value. If one or more values are selected, then those items are used in the filter. Otherwise, whatever is typed into the box is used by the filter.

   For example, you may want to create a filter such as `DEPT.DEPT>5000`, but 5000 is not an existing value in the column. In this case, you can type 5000 into the box above the list of values. The filter will use 5000 as the value.

   If the column is a string, you can type an asterisk at the front or end of the value if you want to use "ends with" or "begins with" wildcard matching.

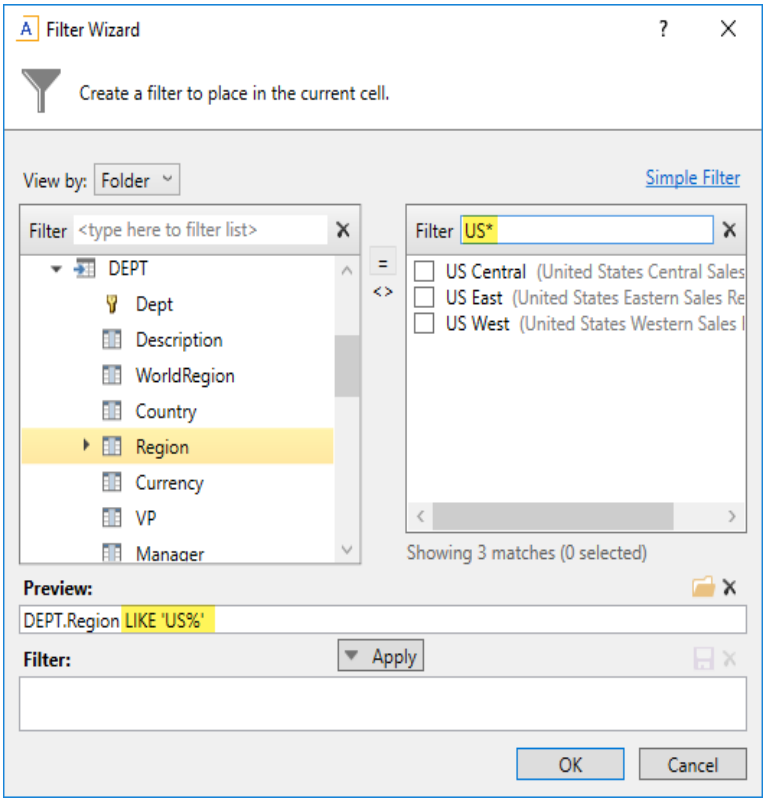
   NOTE: If the selected value is a Date or DateTime value, Axiom will convert the value to standard format if the current locale settings may result in an invalid or inaccurate filter.

3. In the space between the two selection boxes, select the operator to use for the filter criteria statement, such as equals, not equals, greater than, or less than. By default, the filter statement uses equals (=).

   Note the following about filter operators:

   - Greater than / less than options are only available if the column data type holds numbers or dates.
   - If multiple items are selected, then IN and NOT IN syntax is automatically used for equals and not equals respectively. Note that if the operator is equals but you select more items than you have not selected, Axiom will instead use NOT IN syntax for the unselected items to simplify the filter statement.
   - If the column is a string column, and you have typed a value rather than selecting it, then

LIKE and NOT LIKE syntax is automatically used for equals and not equals respectively. By default, wildcard characters (% signs) are placed on both sides of the text, meaning that it will match any value that contains the text. If you place an asterisk to the start or end of the text, then the wildcard character will be only at that location.



Example with wildcards

- If the column is a string column and the value contains an apostrophe (such as O'Connor), the wizard automatically converts this value to double apostrophes so that it is valid for use in the filter (O''Connor). Apostrophes in string values must be escaped this way so that they are not interpreted as the closing apostrophe for the filter criteria statement.

4. Review the filter criteria statement in the **Preview** box to ensure that it is as intended. If you need to make changes, edit your selections made above. The **Preview** box is not editable.
5. Do one of the following:
   - If the filter criteria statement is finished, click **OK**. The filter wizard will use the statement in the **Preview** box (you do not have to click **Apply** in this case).
   - If you want to create a compound filter, click **Apply** to move the current criteria statement into the **Filter** box. Then, repeat steps 1-4 to create another criteria statement. When the next statement is complete, click either **AND** or **OR** to join it to the prior statement.

You can repeat this process as many times as necessary to create the desired statement. You can also edit the full criteria statement within the **Filter** box as needed. When the entire statement is complete, click **OK**.

> **TIP:** If you want to save the filter you have created for future use, click the save icon to the right of the Filter box. You can select a folder location in the Filters Library (or My Documents if applicable), and specify a name for the filter. This option is only available if you have read/write access to at least one location where filters can be stored.

Advanced filter (web version)

Using the web version of the Advanced Filter dialog, you can create a filter using any relevant table and column, and using any supported operator. The web version is used by the following features: the ShowFilterWizardDialog function, the AdvancedFilter refresh variable, and the Filter Wizard command adapter. The web version is also used by the Filter Library when creating or editing a filter directly from the library. The web version of the dialog is the same version that is used in the Web Client.

The web version of the dialog has similar functionality as the desktop version, but there are a few differences. Additionally, the web version of the dialog does not support switching to Hierarchy view.

To create a standard filter using the web Advanced Filter dialog:

1. In the left-hand side of the dialog, select the table column on which you want to base the filter.

   For example, if you want to create a filter such as `DEPT.DEPT>=5000`, then you must select the DEPT column from the DEPT table.

   To find the desired table and column, you can filter the list by typing into the Search box. The filter matches based on table and column names.

   Once you select a table column, the values in that column display in the right-hand side of the dialog.

   > **TIP:** Alternatively, you can use the folder icon to the right of the **Preview** box to load a previously saved filter from the Filters Library. If you do this, your selected filter is placed in the Preview box, overwriting any current content in the preview. Skip to step 4.

2. In the right-hand side of the dialog, select the value(s) on which you want to base the filter.

   You can type into the filter box below the list of values to filter the list. Your current typed value is always placed at the top of the list. You can select this typed value regardless of whether it currently matches an actual value in the column. This behavior is to allow you to create a filter for empty tables, or for tables where the value you want to filter on is not yet present in the column. This is why you may see the "no matches" message but still have one value in the list—your typed value.

3. In the space between the two selection boxes, select the operator to use for the filter criteria

statement, such as equals, not equals, greater than, or less than. By default, the filter statement uses equals (=).

Note the following about filter operators:

- Greater than / less than options are only available if the column data type holds numbers or dates.
- If multiple items are selected, then IN and NOT IN syntax is automatically used for equals and not equals respectively.
- If the column is a string column and the value contains an apostrophe (such as O'Connor), the wizard automatically converts this value to double apostrophes so that it is valid for use in the filter (O''Connor). Apostrophes in string values must be escaped this way so that they are not interpreted as the closing apostrophe for the filter criteria statement.
- The LIKE operator is supported, but is not available for selection in the Filter Wizard. You must manually edit the filter criteria statement if you want to use it. Only advanced users with knowledge of valid SQL LIKE syntax should do this.

4. Review the filter criteria statement in the **Preview** box to ensure that it is as intended. If you need to make changes, you can manually edit the statement, or you can start again with a new statement. If you want to clear the statement, click the **X** icon to the right of the Preview box.

   For more information on valid syntax, see Filter criteria syntax.

5. If no filter is currently present in the **Filter** box, click **Apply** to move the filter down to the Filter box. If a filter is currently present in the Filter box, you can do one of the following:
   - Click **Replace** to overwrite the current filter with the preview filter.
   - Click **AND** or **OR** to add the preview filter to the current filter. This creates a compound criteria statement.

   You can repeat the filter creation process as many times as necessary to create the desired statement. You can also manually modify the filter in the Filter box as needed, such as to add parentheses to group statements.

6. When the filter in the Filter box is complete, click **OK**.

   TIP: If you want to save the filter you have created for future use, click the save icon to the right of the Filter box. You can select a folder location in the Filters Library (or My Documents if applicable), and specify a name for the filter. This option is only available if you have read/write access to at least one location where filters can be stored.

Table and column visibility

Whenever possible, the Filter Wizard is context-sensitive, meaning that it only displays hierarchies and tables that are relevant to the current context. For example, if you are defining a filter for a file group permission set in Security, the Filter Wizard is limited to the plan code table (and any hierarchies defined for that table).

The available tables and columns in the Filter Wizard are also subject to the following settings:

- **Security:** If a user does not have any read access to a table, then that table does not display in the Filter Wizard. If a user has filtered read access to a table, then the filter is applied to the values displayed in the wizard.
- **Column Properties:** Individual columns in a table can be configured to be hidden in the Filter Wizard, using the **Is Filter Column** setting. This may be used to hide columns that are unlikely to be used in filters. Filters can still be manually created using these columns; the properly simply hides the column from the user interface, to streamline the column list.

Using the GetData Function Wizard

You can use the GetData Function Wizard to assist you in building GetData functions. GetData functions can be used in Axiom files to return table data. For example, GetData can be used to:

- Look up department and account descriptions for use in headers and labels in planning files and reports.
- Look up drivers and statistics for use in planning calculations.
- Return individual data points or summed data, such as NYB1 data for all revenue accounts for department 4200.

To use the GetData Function Wizard, right-click in any cell in an Axiom file and then select **Axiom Wizards > GetData Function Wizard**. You can also launch the wizard using the Function Library on the Axiom Designer tab.

In the wizard, you can specify a database column and table, and a filter criteria statement. You cannot complete the other optional parameters of GetData, and you cannot use cell references or Excel functions. However, you can use the wizard to create a basic function, and then edit the function as desired once it has been inserted into the spreadsheet.

The GetData function wizard can only be used to build new GetData functions, it does not edit existing functions. If you open the wizard from a cell that already contains a GetData function (or any formula or value), the contents of the cell will be overwritten with the new GetData function.

As you complete the wizard options, the GetData function is constructed in the **Preview** box at the bottom of the dialog. This box shows you what will be inserted into the cell when you click **OK**.

Selecting a column and table

In the wizard, the only required selection for GetData is the database column. When you select a database column, the table (third parameter) is completed automatically. If the selected column is an alias, the wizard specifies "Alias" as the table.

NOTES:

- To find the desired column, you can view the list by table, table type, folder, and alias. You can also filter the list by typing into the filter box.
- To select an alias, you must set **View by** to **Alias**. This lists all aliases defined in the system. Alias columns are not listed under the tables they are associated with.

Key columns are automatically filtered out of the column list.

Defining a filter criteria statement

To define the filter criteria statement for the function, you can type a statement, or use the Filter Wizard. If you type a statement, you can use the **Validate filter** button to validate the filter syntax.

The filter criteria statement is not required, but it will be used in most cases. If the selected column is from a reference table, the filter criteria statement is necessary to specify the point of data to be returned. If the selected column is from a data table, the filter criteria statement is necessary to specify the rows to be summed, otherwise the function will return summed data for all rows.

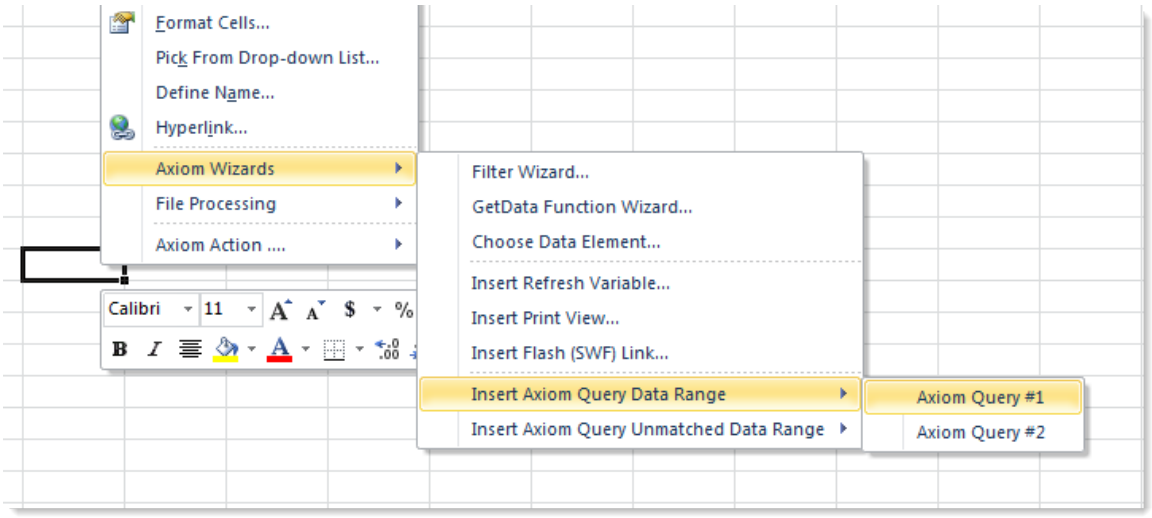
Using Insert Axiom Query Data Range

You can use the **Insert Axiom Query Data Range** tool to define a data range for an Axiom query. This option is only available on the right-click menu if the file has a Control Sheet and if the current sheet is defined on it.

To insert an Axiom query tag:

1. If the query is a vertical query, locate the row in the sheet where you want to start the data range. If the query is a horizontal query, locate the column in the sheet where you want to start the data range.
2. Right-click in any cell of that row or column, and then select **Axiom Wizards > Insert Axiom Query Data Range > *AQName***.

   The *AQName* will either be the default name, such as **Axiom Query #1**, or the name defined for the query on the Control Sheet. The following example shows the default name:



After you select a query, the **Insert Data Range** dialog opens.

3. Do one of the following:
   - If you want to insert the data range tags without a filter, then leave the Filter box blank and click **OK**.
   - If you want to include a filter in the data range, you can either type a filter criteria statement into the Filter box, or use the Filter Wizard . Then, click **OK**.

Axiom finds the defined *insert control column* or *insert control row* for the query, and inserts Axiom query tags in the appropriate column or row, starting in the location where you right-clicked.

For example, if you right-clicked in row 25, and selected **Insert Axiom Query Data Range > Axiom Query #1**, the following tags would be placed in rows 25 and 26, in the designated insert control column for Axiom query #1 (for example, column A):

```
[aq1]
[stop]
```

If you specified a filter for the data range, the tags would be inserted as follows:

```
[aq1;dept.region='northwest']
[stop]
```

Once tags have been inserted, you can double-click the `[aq#]` tag to edit the filter.

NOTES:

- You can also use the **Insert Axiom Query Unmatched Data Range** option to insert a range to check for unmatched data. For more information, see Checking for unmatched data in an Axiom query.
- As an alternative to using this tool, you can use the arrow button to the right of the Axiom query name on the Sheet Assistant. This button inserts a plain set of tags at the current cursor location, in the insert control column or row.

Using Choose Data Element

Using **Choose Data Element**, you can look up data elements from reference tables and then insert the selected element into the current cell.

To look up a data element:

- Right-click in a cell and then select **Axiom Wizards > Choose Data Element**.

To find the desired table, you can view the list by table, table type, and folder. You can filter the list by table class, by selecting or clearing the **Reference** or **Doc. Reference** check boxes. For example, if you clear the **Doc. Reference** check box, then document reference tables do not display in the list.

When you select a table in the left-hand side of the dialog, the values for the table display in the right-hand side of the dialog.

By default, the values in the first column of the table (the key column) display in the dialog. If the table contains a column named Description, then the descriptions display in parentheses next to the values.

If you want to insert a value into the cell, select the value in the list and then click **OK**. Only the key code value is inserted (for example, if the table is ACCT, the account code is inserted).

Enabling or disabling auto-calculation

You may want to change your spreadsheet auto-calculation settings while you are setting up files.

To toggle auto-calculation on or off:

- In the Excel Client: On the **Formulas** tab, in the **Calculation** group, select **Automatic** or **Manual**.
- In the Windows Client: On the **Home** tab, in the **Calculation** group, select **Automatic** or **Manual**.

Also, note the following differences in the calculation options for each client:

- In Excel, you have the option to calculate the current sheet only (**Calculate Sheet**). This option is not available in the Windows Client.
- In the Windows Client, you have the option to calculate the current workbook only (**Calculate Workbook**). This option is not available in Excel.

- Both clients have the option to **Calculate Now**, which calculates all open workbooks.

Removing workbook and worksheet protections

Axiom applies a variety of workbook and worksheet protections to Axiom files. If you have the appropriate user rights, you can temporarily remove the workbook or worksheet protections using the unprotect options available on the Axiom ribbon.

NOTE: The workbook and worksheet protection toggles are intended to allow users to temporarily remove and then reapply the protection settings of the existing file. If the sheet or file is not already configured on the Control Sheet to enable protection, selecting these options does not change that configuration. If you want to enable protection for a sheet or a file, use the Control Sheet settings.

Remove worksheet protection

If a sheet is protected, you can temporarily unprotect it as follows:

- On the **Axiom** tab, in the **Advanced** group, click **Protect > Worksheet** to remove the check mark.

  NOTE: In systems with installed products, this feature may be located on the **Admin** tab—either directly on the ribbon or under **File Protection**.

The sheet can now be edited as desired. If you do not have rights to remove worksheet protection, then this item does not display on the menu.

The sheet protection will be automatically reapplied the next time the file is opened. If you want to reapply it now, click **Protect > Worksheet** again to toggle the protection back on.

The Control Sheet settings for an Axiom file determine whether protection is enabled for a sheet. Sheet protection is most often applied to plan files / templates, but may also be applied to driver files and reports as needed. Protection is always applied to Control Sheets.

Remove workbook protection

If a workbook is protected, you can temporarily unprotect it as follows:

- On the **Axiom** tab, in the **Advanced** group, click **Protect > Workbook** to remove the check mark.

  NOTE: In systems with installed products, this feature may be located on the **Admin** tab—either directly on the ribbon or under **File Protection**.

You can now make structure changes to the workbook, such as unhiding sheets or adding sheets. If you do not have rights to remove workbook protection, then this item does not display on the menu.

The workbook protection will be automatically reapplied the next time the file is opened. If you want to reapply it now, click **Protect > Workbook** again to toggle the protection back on.

For report files and driver files, the Control Sheet settings determine whether workbook protection is enabled for the file. For plan files, workbook protection is applied automatically.

NOTE: For plan files, any structure changes made to the file should be temporary (such as unhiding the Control Sheet to troubleshoot an Axiom query). If you delete or rename sheets, this may impact Axiom query settings and use of calc method libraries. Structure changes to plan files should be made in the template, not the individual plan files.

Remove all protection

Users with the appropriate permissions can turn off all protections and display controls at once by using the **Show Everything** button on the **Axiom Designer** tab.

NOTE: In systems with installed products, this feature may be located on the **Admin** tab—either directly on the ribbon or under **File Protection**.

Adding sheets to Axiom files

Axiom files can contain multiple sheets, and each sheet can be configured on the Control Sheet. To add a sheet to an Axiom file, use standard Excel functionality. (If you are using the Windows Client, use the Workbook Explorer.) The workbook must be unprotected in order to add a sheet.

If you want to configure the sheet to save to the database or use Axiom queries, it may be easiest to copy an existing sheet in the file that uses these features, and then modify the copied sheet as needed instead of starting from scratch.

Once you have added a sheet to an Axiom file, remember to do the following:

- Name the sheet.
- Enter the new sheet name in the first unused column of the Control Sheet.

You can then edit the Control Sheet settings as appropriate for the functionality you want on that sheet. Keep in mind that if you later rename the sheet, you must manually update the sheet name on the Control Sheet as well. The Control Sheet is not automatically updated for changed sheet names.

NOTE: You do not have to add all sheets in the file to the Control Sheet. You only have to add the sheet to the Control Sheet if you intend to use Axiom functionality that depends on Control Sheet settings.

Creating Axiom files from existing spreadsheets

If desired, you can take a standard Microsoft Excel spreadsheet and convert it into an Axiom file by saving it to the Axiom file system and then adding a Control Sheet. Using this process, you can create reports, templates, or driver files.

The spreadsheet file must be XLSX or XLSM format. XLS format files are not supported for Axiom files.

NOTE: Only administrators can add a control sheet to a file and save files to locations other than the Reports Library.

The basic process is as follows:

1. Open the spreadsheet in Axiom.
2. On the **Axiom Designer** tab, in the **Developer** group, click **Tools > Add a Control Sheet > Default**.
3. Configure the Control Sheet as appropriate to meet the intended use of the file.

   For example, if you wanted to use this spreadsheet as a driver file, you would have to complete the settings to enable Save Type 3.
4. Modify the sheet or sheets as necessary to meet the intended use of the file.

   For example, if you are creating a driver file, you would need to ensure that Column A and Rows 1-3 contained the correct information.
5. Use **Save As** to save the file to the appropriate location in the Axiom file system. If the desired location is not available when using Save As, you can instead save the file locally and then use Axiom Explorer to import it into the appropriate location.

File testing and troubleshooting

This section describes the tools available to help test your Axiom files and troubleshoot file issues.

Using file diagnostics for troubleshooting and optimization

Axiom provides a set of file diagnostics that can be used to check your files for potential issues. We recommend using these diagnostics as part of the initial file creation process to help ensure files are set up correctly and optimized, and also on an ongoing basis to make sure no subsequent changes have introduced potential issues.

NOTE: These tests are provided to help you identify issues with your files. Keep in mind the tests are not all-inclusive of every possible design consideration, and in many cases the results are simply a guideline. Files should always be manually tested in their expected environments and for their planned use cases.

Creating a diagnostics task pane or ribbon tab

File diagnostics are run by using the **Run QA Diagnostics** command in a custom task pane or a custom ribbon tab. You can create your own task pane or ribbon tab, or you can use the templates provided by Axiom. The template locations are as follows:

- Task pane: `\Axiom\Axiom System\Document Templates\Sample Task Panes\QA Diagnostics.axl`
- Ribbon tab: `\Axiom\Axiom System\Document Templates\Sample Ribbon Tabs\QA Diagnostics Ribbon.axl`

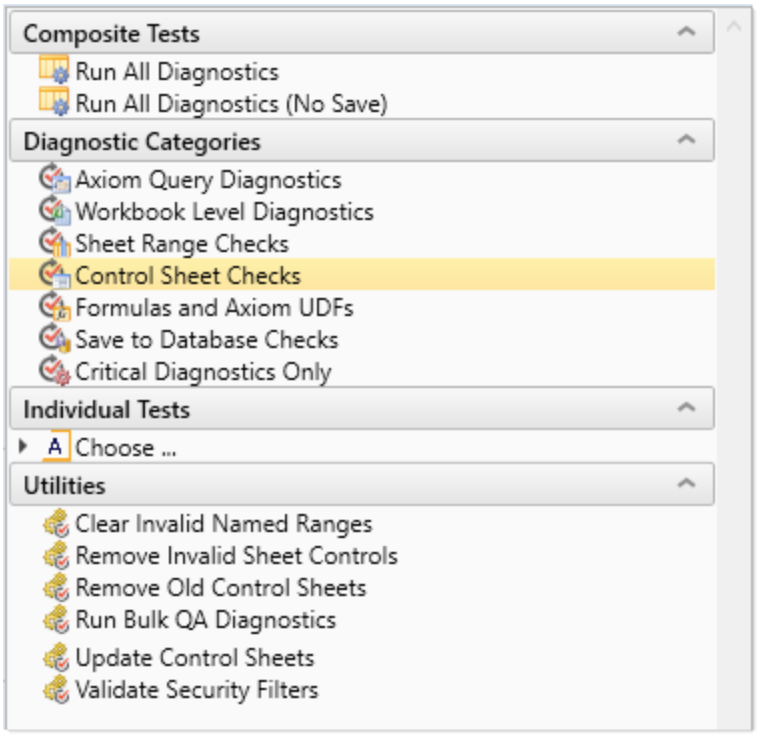
To use a template, an administrator must copy the file to the Task Panes Library or the Ribbon Tabs Library. In order to use the ribbon tab, it must be assigned as a startup file in security. The task pane can be opened directly from the library as needed, or assigned as a startup file.

The template is set up so that you can choose which tests you want to run:

- **Composite Tests**: You can run all diagnostic tests (with or without the save-to-database test).
- **Diagnostics Categories**: You can run all tests in a particular diagnostic category, such as Axiom query tests or Control Sheet tests.
- **Individual Tests**: You can choose individual tests to run.
- **Utilities**: You can run utilities to fix certain issues that are found by the tests.

If you find your own combination of tests that you like to run as a group, then you can modify the template to add your preferred tests (or create your own task pane or ribbon tab). Some tests are always run whenever any diagnostics are run. This includes the log analysis and the performance analysis.

The following screenshot shows an example of the QA Diagnostics task pane template. The ribbon tab template has the same contents.



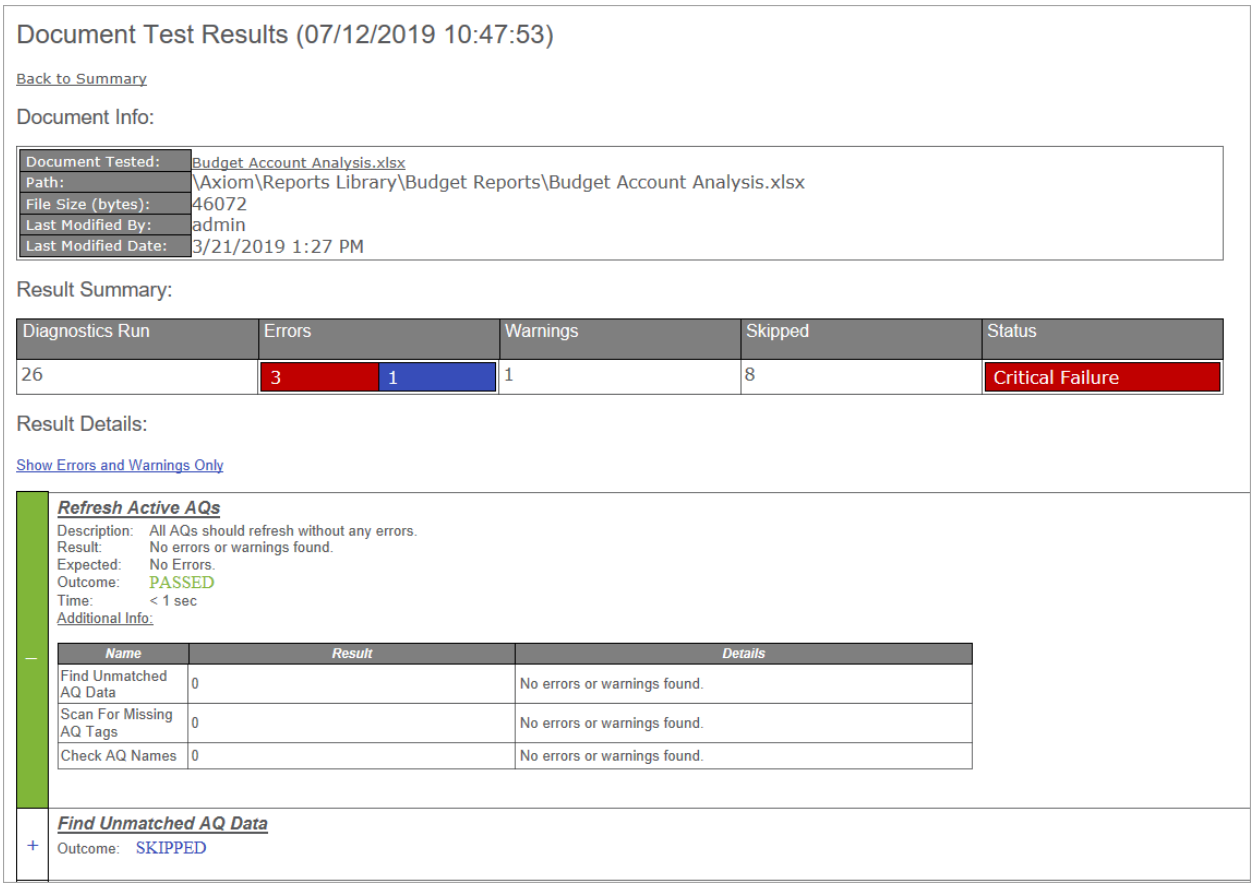
If new tests are added in the future, they will run within the "Run all" tests and within whatever category they are assigned to, but they will not show up in the individual test section. You can manually add the new tests to your existing task pane or ribbon tab, or you can obtain a copy of the updated template from Axiom.

- **To manually add new tests:** Create a new item in your task pane or ribbon tab. In the **Shortcut Target** for the new item, point to the **Run QA Diagnostics** command in the Command Library. Then in the **Shortcut Parameters**, select the test that you want to associate with the item in the task pane or ribbon tab.
- **To obtain an updated template**: The task pane and ribbon tab templates are automatically updated when your system is upgraded. You can copy the updated files from the Document Templates locations listed at the start of this section.

Running diagnostic tests and utilities

Once you have created a QA Diagnostics task pane or ribbon tab, you can run diagnostic tests and utilities on a per file basis. To run diagnostics on a particular Axiom file, open the file and then use the desired item on the task pane or ribbon tab. After running the selected tests, the results display in an HTML page that opens in your browser.

A result summary displays at the top of the page, followed by detailed results for each test. If the results contain any errors or warnings, check the detailed results and address the issues as appropriate. The result format is continually evolving, but appears similar to the following example:



Document Test Results (07/12/2019 10:47:53)

Back to Summary

Document Info:

Document Tested:	Budget Account Analysis.xlsx
Path:	\Axiom\Reports Library\Budget Reports\Budget Account Analysis.xlsx
File Size (bytes):	46072
Last Modified By:	admin
Last Modified Date:	3/21/2019 1:27 PM

Result Summary:

Diagnostics Run	Errors	Warnings	Skipped	Status
26	3 1	1	8	Critical Failure

Result Details:

Show Errors and Warnings Only

Refresh Active AQs

Description: All AQs should refresh without any errors.
Result: No errors or warnings found.
Expected: No Errors.
Outcome: PASSED
Time: < 1 sec
Additional Info:

Name	Result	Details
Find Unmatched AQ Data	0	No errors or warnings found.
Scan For Missing AQ Tags	0	No errors or warnings found.
Check AQ Names	0	No errors or warnings found.

Find Unmatched AQ Data

Outcome: SKIPPED

NOTES:

- Use caution before running any tests that include testing save-to-database. This includes **Run All Diagnostics**, the **Save to Database Check** category, and the **Check for Save Errors** individual test. Running the save-to-database test will save the current file and perform a save-to-database. Do not run this test if you do not want to save the file and its data.
- If you have made any changes to the file, it is a good idea to save it before running the diagnostic tests. Depending on which tests you are performing, they may perform actions such as refreshing Axiom queries, which will impact the contents of the file.
- No particular permissions are required to use the Run QA Diagnostics command, but individual tests may require certain permissions in order to execute. If the user does not have those permissions, the test will fail. For example, the save-to-database test requires the user to have read/write access and Allow Save Data.

Testing files in bulk

You can use the **Run Bulk QA Diagnostics** utility to run tests on all files in a particular folder. This utility is available in the **Utilities** section of the standard task pane or ribbon tab template, or you can add it to your own task pane or ribbon tab.

When you use this utility, you are first prompted to select a folder. The diagnostics tests will be run on all eligible spreadsheet files in that folder and its subfolders. All tests will be run except for tests that require saving the file or performing a save-to-database.

When the process is complete, a summary page opens in the browser. This page indicates the errors and warnings in each processed file. You can click on a file name to be taken to the full results for that file.

NOTE: Some tests can be run on the following additional file types: Scheduler jobs, imports, exports, task panes, and ribbon tabs. For this reason, you can also select folders in the corresponding libraries for these file types. When executing tests on these folders, only the eligible tests will be run.

Diagnostic tests

The following diagnostic tests are available:

Test	Description
Analyze Axiom UDFs	Analyzes all Axiom formulas in the file with the following goals: • Reports the total number of Axiom formulas found. A large number of formulas may impact performance. • Reports any warning conditions for certain formulas.
Analyze Excel Formulas	Analyzes all Excel formulas in the file with the following goals: • Reports the total number of Excel formulas found. A large number of formulas may impact performance. • Reports any warning conditions for certain formulas—for example, warning about settings that may not be compatible with the Windows Client.
Cell Formula Error Count	Reports the total number of cell errors in the file, and lists errors found. This test checks for both Excel formula errors and Axiom formula errors (#ERR).
Check AQ Names	Checks to see whether all active Axiom queries have defined names (rather than the default "Axiom Query #1", etc.). This test is also included in the **Refresh All Active AQs** test.
Check Data Validation	Checks for specific uses of data validation that may be incompatible with the Windows Client.

Test	Description
Check Document Links	Checks for invalid document references used in the file, such as an invalid path or file name used in the Associated Task Pane field. **NOTE:** In addition to Axiom spreadsheet files, this test can be run on Scheduler jobs, imports, exports, task panes, and ribbon tabs. To run the test on these additional file types, use the **Run Bulk QA Diagnostics** utility.
Check Embedded Images	Checks any embedded images for potential issues, such as use of file types that are incompatible with the Windows Client, or very large file sizes that may cause performance issues.
Check File Size	Reports the size of the file. Larger files may have worse performance, and may indicate an overall design issue.
Check for External Links	Checks the workbook for links to external files, and issues a warning if found. External links should be avoided whenever possible to help prevent the possibility of broken links.
Check Formula Lengths	Checks the length of all formulas in the file and issues a warning or an error if a formula exceeds a certain threshold of character length. Lengthy formulas may indicate an overly complicated design that could potentially be simplified.
Check for Old Control Sheet	Checks the Control Sheet of the workbook to see if it is out of date and should be upgraded to the latest.
Check for Save Errors	Saves the file, including a save-to-database if applicable, and reports any save errors. **NOTE:** Use caution before enabling this test. Do not include this test if you do not want to save the file and its data.
Check for VBA	Checks whether VBA is present in the file. Files should not contain VBA unless it is absolutely necessary to support a known custom solution.
Check Named Ranges	Checks all named ranges in the file to be sure they refer to valid worksheet ranges. Invalid named ranges can cause errors when performing certain Axiom operations, and should be corrected or removed as appropriate.
Circular Reference Check	Checks for circular references in the file and warns if found. Although in some cases circular references may be valid and intentional, in other cases they are likely to be the result of a design error and should be corrected. Circular references can cause issues with GetData calculations.
Conditional Format Count	Reports the total number of conditional formats used in the file. A large number of conditional formats may impact performance, and certain configurations may be incompatible with the Windows Client.

Test	Description
Find Invalid Sheet Controls	Checks for invalid sheet names on the Control Sheet, as well as the presence of old Control Sheets (left over after upgrading).
Find Invalid Sheet Filters	Checks for invalid sheet filters on the Control Sheet.
Find Unmatched AQ Data	Checks for unmatched data for all Axiom queries in the file. Data is unmatched if it is brought back by the database query but it is not inserted or updated within any data ranges. You may want to check for unmatched data for performance reasons, or because you intended for all data to be matched. This test is also included in the **Refresh All Active AQs** test.
GetData Diagnostics	Analyzes all GetData formulas in the file and returns information on the server load required to resolve them, as well as any errors.
Hidden Formula Scan	Checks for hidden formulas, meaning formulas in cells with the **Hidden** protection option enabled. Hidden formulas should be avoided as they may cause issues with Axiom processes, such as action codes and views.
Max Column Number	Reports the total number of columns in the widest used range in the file. If the used range is larger than expected, you should reduce it to the actual used area to improve performance.
Max Row Number	Reports the total number of rows in the tallest used range in the file. If the used range is larger than expected, you should reduce it to the actual used area to improve performance.
Merged Cell Scan	Checks for use of merged cells and issues a warning if found. Merged cells may cause issues with processes that copy and paste, such as action codes and snapshot.
Refresh Active AQs	Refreshes all active Axiom queries in the file and reports any errors found. Inactive queries are not run.
Save2DB Tag Check	Checks for potential issues with save-to-database tags, such as when tags are present but the save-to-database process is not enabled in the sheet. Also checks for hard-coded table names in file groups (should use variables).
Scan for Missing AQ Tags	Checks to see whether all active Axiom queries have corresponding data range tags in the workbook. This test is also included in the **Refresh All Active AQs** test.
Sheet Name Validation	Checks sheet names in the file and reports any potentially problematic names. Remember to update the Control Sheet if you change the name of a sheet.
Total Worksheets	Reports the total number of worksheets in the file. An extremely large number of worksheets may indicate an overall design issue.

Test	Description
Validate AQ Refresh Settings	Checks for common design issues with the Axiom query refresh settings.

Diagnostic utilities

The following utilities are available to help resolve issues found by the file diagnostics. You can run these utilities on a single file by opening that file and then executing the utility, or you can run these utilities on all files in a designated folder. In order to run a utility on a file, you must have Sheet Assistant permission to that file.

The folder-level functionality is only available to administrators, and only when the currently active file is *not* a managed spreadsheet file (otherwise, the utility will be performed on that managed spreadsheet file). When you execute the utility, you will be prompted to select a folder. For an example of how to execute a utility on a folder, see Updating a Control Sheet.

Utility	Description
Clear Invalid Named Ranges	Removes any invalid named ranges from the file.
Remove Invalid Sheet Controls	Removes any invalid sheet controls from the Control Sheet. An invalid sheet control is a sheet name that is listed on the Control Sheet but does not correspond to an actual sheet in the file. The utility removes the entire column from the Control Sheet. **NOTE:** If the sheet control is invalid because you have renamed a sheet, then you should rename the entry on the Control Sheet rather than running this utility.
Remove Old Control Sheets	Removes any old Control Sheets in the file. These are archived Control Sheets that result from performing a Control Sheet upgrade. The sheet names of these Control Sheets are prefixed with "Old_".
Update Control Sheets	Updates the Control Sheet in the file to the latest version. This is the same process that is performed by the **Update the Control Sheet** command in the Sheet Assistant.

The following additional utilities are available to administrators:

Utility	Description
Run Bulk QA Diagnostics	Run QA Diagnostics tests on all files in a selected folder and its subfolders. All tests will be run except for tests that require saving the file or performing a save-to-database.

Utility	Description
Validate Security Filters	Validates filters set in security, including plan file filters, table type filters, and table filters. The utility uses the same validation routine as the Security Management dialog. Filters defined on users, roles, and subsystems are validated. It may be useful to run this utility after changing security filters using Open Security in Spreadsheet or Save Type 4. For performance reasons, only a limited number of filters are validated when using these features. **NOTE:** This utility does not run against the current file, it runs against the database.

Using Developer Mode

When you are designing or troubleshooting Axiom files, you can place your session in Developer Mode. Developer Mode does the following:

- Enables additional validation for certain features, to help catch issues during file design
- Provides access to analysis tools in the Developer Mode task pane, to help test and troubleshoot the file

Developer Mode is enabled or disabled using the Axiom Designer task pane. By default, the Axiom Designer task pane is only available to administrators, but this can be customized so that other users can access the task pane. Additionally, you can place the Developer Mode option on a custom task pane or ribbon tab by using the **Developer Mode** command, and that task pane or ribbon tab can be made available to any user. The Developer Mode command itself is not restricted by any security permissions; if a user has access to the command, they can enable and disable Developer Mode for their session.

To toggle Developer Mode on or off:

- On the **Axiom Designer** tab, in the **Developer** group, select or clear the **Developer Mode** check box.

By default, Developer Mode is disabled. If you enable Developer Mode, the features summarized above become available and the Developer task pane displays in the Axiom Assistant area. (This only impacts your current session; other user sessions are not impacted.) While Developer Mode is enabled, the text **(DEVELOPER MODE)** is appended to all Axiom-related ribbon tabs.

Developer Mode is automatically disabled if you close and reopen the system. The enabled state is not remembered and must be re-enabled the next time you log in.

Developer Mode behavior

When Developer Mode is enabled, the following behavior applies:

- The Developer task pane becomes available.
- Sheet filters are validated prior to submitting data queries to the database, to provide more detailed error messages to file designers about invalid sheet filters. When Developer Mode is disabled, sheet filters are not separately validated for performance reasons. If a sheet filter is invalid when Developer Mode is disabled, the invalid syntax is submitted to the database and may result in a SQL error.

Developer Mode task pane

The Developer Mode task pane automatically shows and hides based on whether Developer Mode is currently enabled or disabled. The task pane cannot be manually opened or closed, and it cannot be customized.

Currently, the Developer Mode task pane contains tools and statistics that are only useful for software developers or very advanced file designers. The task pane records information such as:

- Number of calls made to the server
- Amount of data sent to the server and received from the server
- Number of Get Data Calls and New Items

The following options are available in the Developer Mode task pane:

Item	Description
Reset	Click this button to clear the statistics shown in the task pane.
Enabled	Specifies whether the task pane gathers statistics. If disabled, the task pane will not record activity.
Collect Call Stacks	Specifies whether call stack information is shown in the task pane. If disabled, the task pane will not show call stack information.

Checking a sheet for formula errors

Using the **Find Formula Errors** tool, you can check a sheet for Excel formula errors such as `#REF!`, `#VALUE`, and `#NAME?`.

If a sheet has formula errors, then data cannot be saved to the database from that sheet. You can use this tool to check for errors before saving, and correct them. You can also use this tool when setting up reports, templates, or drivers, to make sure you have not accidentally introduced a formula error while editing the file.

Only standard Excel formula errors are found by the tool. Axiom function errors (`#ERR`) are not found by the tool, and do not prevent saving data to the database.

To check a sheet for Excel formula errors:

- On the **Axiom Designer** tab, in the **Developer** group, click **Tools > Find Formula Errors**.

The tool checks the current sheet for formula errors. If errors are found, they are displayed in the **Save Data Errors** task pane that opens in the Axiom Assistant area. This task pane lists the cell where the error was found (if applicable), and a brief description of the error. If the error is associated with a specific cell, then you can double click on the error to be taken directly to the applicable cell in the spreadsheet.

If no formula errors are found, a message box informs you of this.

NOTE: Only administrators have access to this feature in the Axiom Designer tab, however, any user can check for formula errors by running file diagnostics. For more information, see Using file diagnostics for troubleshooting and optimization.

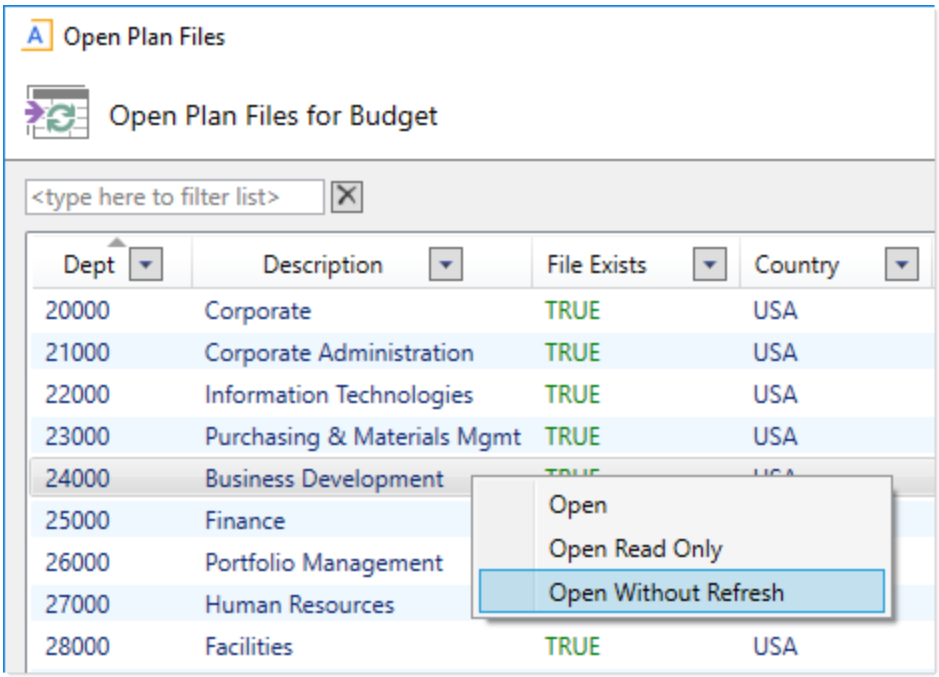
Opening an Axiom file without refreshing data

When troubleshooting Axiom spreadsheet files, it can be helpful to open the file without performing any data queries. This allows you to review the file in its initial state, before any queries are run. It also allows you to open the file quickly, especially in cases where the file is configured to run many queries on open (or when one or more of the queries is time-consuming to run).

To open an Axiom spreadsheet file without refreshing data, use the **Open Without Refresh** feature. This feature is available on the right-click menu in the following locations:

- **Axiom Explorer** dialog
- **Explorer** task pane
- **Open Plan Files** dialog
- **Reports Library** dialog

NOTE: Open Without Refresh is only available to administrators.



Example Open Without Refresh option in the Open Plan Files dialog

When you open a file using Open Without Refresh, the following "file open" processes do *not* occur:

- Axiom queries set to **Refresh on file open** are not run.
- Data lookups listed in **DataLookups to run on open** are not run. However, data lookups using the reserved name **AxRefreshOnOpen** are still run.
- The option **Refresh all Axiom functions on open** is ignored (functions are not refreshed).
- If the file is configured to show a refresh dialog on open (using **Refresh Forms Run Behavior**), the refresh dialog does not display.
- The option **Downgrade to read-only on open** is ignored.

Control Sheet Settings

The Control Sheet contains a column of settings in the left-hand side of the sheet, followed by a series of definition columns for sheets in the workbook. Each sheet where you want to use Axiom file functionality must be configured on the Control Sheet.

To configure a sheet on the Control Sheet, go to the first open column on the Control Sheet and enter the sheet name into row 1. If you ever change the sheet name, you will need to update this cell on the Control Sheet. You can then complete the settings for the sheet as desired.

The definition columns do not need to be contiguous, and the order does not matter. Control Sheet settings are applied to any sheet that is listed in row 1. If you later remove a sheet from the workbook, you can simply delete the sheet name from row 1. You do not need to delete the definition column.

The Control Sheet uses spreadsheet grouping to group related sections. To view the settings within a section, click the plus icon in the left-hand sidebar. To close a section, click the minus icon. You can also expand or collapse groupings by clicking on the number icons at the top of the left-hand sidebar (clicking 1 collapses all sections, and clicking 3 opens all sections).

NOTE: If your Control Sheet does not contain a setting listed here, then it is out of date. If you want to use a setting that does not exist on the current Control Sheet, you must update the Control Sheet using the Sheet Assistant. For more information, see Updating a Control Sheet.

IMPORTANT: It is not recommended to place any non-standard content on the Control Sheet itself. When Control Sheets are upgraded, only the standard settings are migrated to the new sheet. If you have values that you want to use on the Control Sheet but you want to maintain and reference them in one location, you should store these values on another sheet in the file. These cross-sheet references will be automatically maintained when you upgrade the Control Sheet.

The tab name of the Control Sheet is Control_Sheet.

Save to Database Setup

Use the save-to-database settings to configure a sheet to save to the database. This is most typically used for templates/plan files and driver files, but it can also be used in report files. For more information on saving to the database and the various save types, see Overview of save types.

The save-to-database settings are protected on the Control Sheet. You must unprotect the sheet using **Advanced > Protect > Worksheet** before you can configure a save-to-database process.

NOTE: Make sure you understand how a particular save type works before enabling it. Certain save type settings may overwrite existing data or existing tables.

Settings for Save Type 1

Item	Description
Save Type 1 Enabled	Specifies whether Save Type 1 is enabled for the sheet (**On**/**Off**). Save Type 1 is a flexible save type typically used for templates/plan files and reports. For more information, see Using Save Type 1.
Zero data on save enabled	Specifies whether unmatched data in the table is zeroed before saving, based on a zero tag. In most cases, this should be set to **On**. However, the default setting is **Off** because the destination table must contain a manually created column to hold the zero tags. The name of this column depends on whether you are using the default save behavior (SaveTagDocID) or whether you are using a custom zero tag and column (SaveTagCustom, or a different custom name). The default behavior is used unless you set **Disable DocID During Zero** to **On**. Once you have confirmed that the destination table contains the required column, you can enable this setting. All of the remaining Save Type 1 settings in this section only apply when this setting is **On**. **NOTE:** If the Save Type 1 process is for an Axiom form (or for any file that is intended to be used as read-only with save data), then use caution before enabling the zero on save option. The data saved by an Axiom form is not persisted in the file, therefore the data saved by one user may be totally different than the data saved by another user, depending on the configuration of the Axiom form. If zero on save is enabled in this circumstance, the second user would zero the data saved by the first user. For more information, see: • How Save Type 1 works • Using zero tags with Save Type 1

Item	Description
(advanced) Disable DocID During Zero	Specifies whether the default SaveTagDocID column is used when zeroing data on save. This setting only applies if **Zero data on save enabled** is set to **On**. • If **Off** (the default), then the document ID is saved to the SaveTagDocID column when data is saved. This column must exist in the target table, or else an error will occur when saving. • If **On**, then the default behavior is disabled. In this case, you must specify a **ZeroTagValue** for the save process, and you must either specify a **CustomZeroTag Column**, or use the default custom column of SaveTagCustom. This setting does not have to be set to On in order to use a custom zero tag and column. If this setting is **Off** and a custom zero tag and column are defined, then both columns will be used. **NOTE:** This setting can also be defined within the Save2DB tag for a particular save process. If it is defined there, then the Control Sheet setting is ignored. For more information, see Using zero tags with Save Type 1.
(advanced) ZeroTagValue	Defines a custom zero tag for the zero process performed during a save-to-database. If you use this option, then you must also define a **CustomZeroTagColumn** (or use the default custom column of SaveTagCustom). **NOTE:** This setting can also be defined within the Save2DB tag for a particular save process. If it is defined there, then the Control Sheet setting is ignored. For more information, see Using zero tags with Save Type 1.
(advanced) CustomZeroTag Column	Specifies a column to hold the custom ZeroTagValue for the zero process. If no column name is specified here but the custom ZeroTagValue is being used, Axiom attempts to use a column named `SaveTagCustom` to hold the custom zero tag. In either case, this column is not automatically added to the destination table; you must create it manually. **NOTE:** This setting can also be defined within the Save2DB tag for a particular save process. If it is defined there, then the Control Sheet setting is ignored. For more information, see Using zero tags with Save Type 1.

Settings for Save Type 3

Item	Description
Save Type 3 Enabled	Specifies whether Save Type 3 is enabled for the sheet (**On**/**Off**). Save Type 3 is a structured save type used for driver files. It can also be used in other Axiom files. If Save Type 3 is enabled for a sheet, no other save types can be enabled for that sheet. For more information, see Using Save Type 3.
Table Name for Save Type 3	Defines the table name for Save Type 3. The name must be unique—no other sheets in this file or any other file can use the same table name.
Table Folder for Save Type 3	Specifies the target virtual folder in the Table Library. If the folder does not already exist, it will be created. When entering the folder location do not place a backward slash in front of the first folder name. For example, enter `Drivers` or `BudgetData\Drivers`, but do not enter `\Drivers`.

Settings for Save Type 4

Item	Description
Save Type 4 Enabled	Specifies whether Save Type 4 is enabled for the sheet (**On**/**Off**). Save Type 4 is used to make database structure changes and edits to system tables. For more information, see Using Save Type 4.

▶ Workbook Options

Use the following options to configure workbook settings for the file.

Item	Description
Workbook Protection On/Off	Determines whether workbook protection is enabled or not. • If enabled, workbook protection is applied when the file is opened, and users cannot add / remove sheets or hide / unhide sheets. • If not enabled, workbook protection is not applied. **NOTE:** Workbook protection is always applied to plan files, regardless of whether this setting is enabled or not. For more information, see Configuring workbook protection for Axiom files.

Item	Description
Workbook Protection On/Off during snapshot	Determines whether workbook protection is applied when a snapshot copy is taken. By default, this option uses a formula to inherit the setting for the “live file” option above. For more information, see Applying protection to snapshot copies.
Downgrade to read-only on open	Determines whether the file is read-only, regardless of the user's security permissions to the file. By default, this is set to **Off**, meaning the user's security permissions determine the access level for the file. This setting is evaluated after the file has been opened and all initial processing and calculations have occurred, including any Axiom queries set to refresh on open. If **On**, then the file is immediately converted to read-only and the user's lock on the file is released. The file tab is updated to reflect the read-only status. (If the file is already open read-only, then no additional action occurs.) You might use this setting if you want to determine whether the file is editable based on something other than the user's security settings. For example, perhaps the file should not be editable at certain periods of the year; in this case you could use a formula to dynamically determine whether the file should be editable or not. This would be easier than manually changing security permissions at certain times of the year. **NOTES:** • This setting should not be used to facilitate read-only, save-to-database reports. Instead, users should be explicitly granted read-only access to the file with Allow Save Data (in Security). If a user has read/write access with Allow Save Data, then the user can only save data from the file when it is opened as read/write. If the file is artificially downgraded to read-only, the read/write user cannot save data. The exception to this rule is when the file is restricted to read-only access because it is owned by an installed product package. • This setting also applies to administrators. If this setting is enabled but an administrator needs to open the file with read/write permissions, use the **Open Without Refresh** feature. For more information, see Opening an Axiom file without refreshing data.

Item	Description
Close read-only files without prompting to save	Determines whether users are prompted to save the file if the file is open read-only and changes are made to it. • If **Off** (default), then users are prompted to save the file if changes are made to it. In this case the user must save the file as another name and/or in a different location in order to save the changes. • If **On**, then users are not prompted to save the file if changes are made to it. This is intended for configurations where end users need to open the file and refresh the data coming into it, but they never need to save the file. The file is simply a vehicle to display data. In this case eliminating the unnecessary save prompt can improve the user experience. This setting applies whenever the file is opened read-only. If the file is opened read/write, users are always prompted to save the file if changes are made.
Process alerts on save data OR Process alerts on save document	These settings determine whether alerts are processed when a save-to-database is performed or when the document is saved. By default both settings are disabled (**Off**). You can choose to enable one or the other of these options, but not both. If both settings are enabled then only the save-to-database alert processing will occur. To enable alert processing for either the save-to-database process or the document save, select one of the following options: • **Process**: When alerts are processed, the normal alert processing results dialog is shown. This is typically only used when the alert author is testing the alert setup for the file. • **ProcessSilently**: Alerts are processed silently in the background; the user performing the save will not be aware of the alert processing (unless, of course, the user receives an alert as a result of the processing). This is the intended setting for production-ready files. If no alerts are defined in the document, then the alert processing is ignored and no error message is displayed. For more information on processing alerts, see the *System Administration Guide*.

Item	Description
Process cross sheet AQ Batches	Specifies whether Axiom query batches are processed across sheets or per sheet. By default this setting is **Off**, which means that any batched Axiom queries are processed per sheet. If set to **On**, then batched Axiom queries are processed across sheets. This means that queries from different sheets can be processed concurrently in the same batch. For more information, see Batch processing for Axiom queries.
Enable parallel save data	Specifies whether parallel processing for save-to-database is enabled. By default this setting is **On**, which means that save blocks in a file are processed concurrently instead of sequentially. This behavior automatically handles potential dependencies and waits for processes to complete as needed. If set to **Off**, then save blocks are processed sequentially, one by one. This may impact performance in files with many save blocks, or with multiple save blocks that process many records. For more information, see How Save Type 1 works.

Item	Description

Associated Task Pane	Specifies a task pane to open automatically when this file is opened. The task pane cannot be closed, and will remain linked to this file (meaning it will show or hide depending on whether this file is the active file). When this file is closed, the task pane is automatically closed.
	
	Specify the full path and file name to one of the following:
	• A custom task pane file in the Task Panes Library (AXL file)
	• A form-enabled file, where the form has been designed for use as a task pane
	
	For example:
	`\Axiom\Task Panes Library\MyTaskPane.axl`
	
	`\Axiom\Reports Library\Forms\MyTaskPane.xlsx`
	
	To easily obtain the full path for a file, navigate to that file in the Explorer task pane, then right-click and select **Copy document path to clipboard**. You can then paste the value into the cell.
	
	Alternatively, you can use document shortcut syntax to specify the task pane file. When using a document shortcut, you can add a parameter to specify an alternate tab name for the task pane. If specified, this tab name will be used instead of the file name. For example:
	`document://\Axiom\Task Panes Library\MyTaskPane.axl?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.
	
	The next time you open the document after saving, the 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, or to change the tab name, simply enter the path or document shortcut as you would have originally, and it will be converted again when you save the file.

Activate sheet on open	Specifies a sheet to set as the active sheet when the file is opened. Enter the name of any visible sheet in the file. This setting can be used so that the file always opens to the specified sheet, regardless of which sheet was active when the file was last saved. For more information, see Setting the active sheet and active cell.

Item	Description
Disable quick filter	Determines whether the Quick Filter feature is available for the report. By default this is set to **Off**, which means that Quick Filter is available. If the report is not designed to be used with Quick Filter, then you can set this to **On**. The Quick Filter functionality will not be available when the report is active. For more information, see Setup considerations for using Quick Filter in a report.
Master Sheets	Specifies one or more sheets in the file as a master sheet, so that end users can add new sheets on-the-fly by copying the designated master sheets. Separate multiple sheet names with commas. This feature applies to all Axiom spreadsheet files except driver files. For more information, see .
Show row and column headings	Specifies whether the spreadsheet row and column headings are shown on each sheet in the workbook. This property can be set as follows: • **<Blank>**: Row and column headings are visible if the file is opened read/write, and hidden if the file is opened read-only. This is the default setting. • **On**: Row and column headings are visible when the file is opened. • **Off**: Row and column headings are hidden when the file is opened. This property only determines the starting point of the headings in the workbook. Users can toggle the headings on and off by using the **Headings** option on the Axiom ribbon tab. For more information, see Configuring default display options for a sheet.
DataLookups to run on open	Specifies one or more data lookup queries to run when the file is opened, by listing the names of the DataLookup data sources. Separate multiple data source names with commas. Data sources will be executed in the order they are listed. The list of data source names can be qualified with sheet names (such as `Sheet1!Data`) or unqualified (just `Data`). If qualified, then Axiom will only scan the listed sheets for the data sources. If unqualified, then Axiom must scan all sheets for the data sources, which can impact performance. All items in the list must be either qualified or unqualified. Mixing qualified and unqualified names will result in an error when the data sources are executed. To run unnamed DataLookup data sources on open, use the keyword `[unnamed]` (either with a sheet name, or without to scan all sheets). The keyword can be used by itself or as part of the comma-separated list of names. For more information, see Data Lookups.

Item	Description
Enable Message Stream	Specifies whether the message stream is enabled for the file, so that users can view comments and add comments about the file. • If set to **On**, the Message Stream task pane will be available when the file is opened in the Desktop Client, and the Message Stream panel will be available when the file is opened in the Web Client (for form-enabled files). • If set to **Off**, the message stream will not be available for the file. Users cannot view or add comments. If comments already exist when the message stream is disabled, they will be retained in the database. If the message stream is re-enabled, the comments will be available again. This setting is enabled by default for all new files created in version 2016.1 or later. For existing files created prior to version 2016.1, the setting is not present and treated as disabled. When you upgrade the Control Sheet of these older files, the setting will be migrated as blank and treated as disabled. You must explicitly turn the setting On after upgrading the Control Sheet, if you want the message stream to be available for these older files.
Clear DataLookups on save	Specifies whether data lookup queries are cleared when the file is saved. By default, this is set to **Off**, which means that any results that are present in the file when it is saved will be retained. This feature is intended for data security. If data lookups are not configured to automatically run when the file is opened, then the data queried and saved by one user will still be present when another user opens the file. This may result in users seeing data outside of their data filter, because the data will not update until the data lookups are executed. If this option is set to **On**, then the result column for all DataLookup data sources will be cleared when the file is saved. This means that users will not see this data until the data lookups are executed. This setting is primarily intended for reports.
Hide Control Sheet on open	Specifies whether the Control Sheet is hidden for all users when the file is opened. By default, this is set to **Off**, which means the user's security permissions determine whether the Control Sheet is hidden by default. If the user is an administrator or has the Sheet Assistant permission, then the Control Sheet is visible by default, otherwise it is hidden. If set to **On**, the Control Sheet is hidden by default for all users. For more information, see Viewing the Control Sheet.

Item	Description
Data Context	Defines a data context name, for use by Axiom forms to control data saves. If a name is defined, then only one user at a time can "lock" the data context and save data from the form. For more information, see the *Axiom Forms and Dashboards Guide*. This option has no effect on spreadsheet Axiom files; it only applies to form-enabled files and only when the file is opened as a form.
Data Context Description	Defines an optional description for the data context, if a data context is defined. The description is displayed on the Save Lock dialog within the Axiom form.
Is Data Context Checked Out	Specifies whether the current user has the save lock, if a data context is defined. This field is system-managed; Axiom automatically populates the field as **On** or **Off** when the file is opened as a form and the file has a defined data context. You can reference this field as needed to dynamically configure form components based on whether the current user has the save lock.

Sheet Options

Use the following options to set visibility, protection, and display settings for the sheet.

Item	Description
Sheet Visible / Hidden	Specifies the visibility settings for the sheet. Available options are: • **Visible** – The sheet is visible to all users. • **Hidden** – The sheet is hidden by default. Sheet visibility settings are applied when the file is opened. For more information, see Configuring sheet visibility for Axiom files.
Sheet Protection On/Off	Determines whether sheet protection is enabled or not. • If enabled, sheet protection is applied *when the file is opened*, and users are restricted to making edits in unprotected cells. • If not enabled, sheet protection is not applied (however, any existing protection is not removed). For more information, see Configuring worksheet protection for Axiom files.
Sheet Protection On/Off during snapshot	Determines whether worksheet protection is applied when a snapshot copy is taken. By default, this option uses a formula to inherit the setting for the "live file" option above. For more information, see Applying protection to snapshot copies.

Item	Description
Freeze Panes	Defines the freeze panes settings for the sheet, using the following syntax: `TopLeftCell:BottomRightCell` For example, `A1:H13` would freeze rows 1 through 12 and columns A through G. Freeze panes settings are applied when the file is first opened, and then reapplied after applying a view. For more information, see Configuring default display options for a sheet.
Active Cell	Specifies the active cell on the sheet when the file is first opened. The sheet will be scrolled so that the specified cell is at the upper-left of the sheet, within the boundary of the freeze panes. **NOTE:** If an Initial Dynamic View is specified for the sheet, and that view has an active cell defined, that cell will be the active cell instead of the cell specified here.
Axiom Double-Click	Specifies whether special Axiom double-click actions are enabled for the sheet: • If enabled, then when you double-click a cell in the sheet, the appropriate Axiom double-click action will be performed (for example, drilling the data or opening a calc method form). If multiple double-click actions could apply to the cell, the highest-priority action is performed. • If disabled (default), then no special Axiom double-click actions will apply to the sheet. For more information, see Double-click behavior.
Enable Drilling	Specifies whether drilling is enabled for the sheet: • If **On** (default), then the **Drill** button on the ribbon is active when this sheet is active. Double-click drilling, if enabled, is also available for the sheet. • If **Off**, then the **Drill** button on the ribbon is disabled when this sheet is active, regardless of whether drillable content exists on the sheet. Double-click drilling is also prevented. For more information, see Disabling drilling for a sheet.
Master Sheet	Displays the name of the master sheet that was used to create this sheet. This field is system-managed; Axiom places the appropriate sheet name here when the sheet is inserted into the file using the **Add New Sheet** command.

Item	Description
Show Gridlines	Specifies whether gridlines are visible in the sheet by default: • **<Blank>**: Gridlines are left as is when the file is opened. The visibility of gridlines is controlled by Excel functionality. • **On**: Gridlines are visible when the file is opened. • **Off**: Gridlines are hidden when the file is opened. For more information, see Configuring default display options for a sheet.
Default Zoom	Specifies the default zoom level of the sheet. If defined, the zoom level is applied when the file is opened. Enter the zoom level as an integer, with or without a percent sign (`90` or `90%`). If blank, the zoom level in the sheet is left as is. The zoom level is controlled by Excel functionality. For more information, see Configuring default display options for a sheet.

Views

Use the following properties to set view options for the sheet.

Item	Description
Initial Dynamic View	Specifies a default sheet view for the sheet. Only a single sheet view name can be specified. This sheet view is applied when the sheet is first made active after the file is opened. For more information, see Defining sheet views for a sheet and Setting the default views for a sheet.
Initial Dynamic Column Views	Specifies one or more default column views for the sheet. Multiple column view names can be specified, separated by commas. The column views are applied when the sheet is first made active after the file is opened. For more information, see Defining column views in a sheet and Setting the default views for a sheet.

Item	Description
Initial Dynamic Row Views	Specifies one or more default row views for the sheet. Multiple row view names can be specified, separated by commas. The row views are applied when the sheet is first made active after the file is opened. For more information, see Defining row views in a sheet and Setting the default views for a sheet.
Current Dynamic View	Displays the name of the currently active sheet view (if applicable). This field is system-managed, and is used to re-apply the sheet view after certain activities performed in the sheet (such as running an Axiom query). If you input any text in this cell, it will be overwritten the next time a view is applied.
Current Dynamic Column Views	Displays the names of the currently active column views (if applicable). Multiple column view names are separated by commas. This field is system-managed, and is used to re-apply the column views after certain activities performed in the sheet (such as running an Axiom query). If you input any text in this cell, it will be overwritten the next time a column view is applied.
Current Dynamic Row Views	Displays the names of the currently active row views (if applicable). Multiple row view names are separated by commas. This field is system-managed, and is used to re-apply the row views after certain activities performed in the sheet (such as running an Axiom query). If you input any text in this cell, it will be overwritten the next time a column view is applied.

Data/Zero Options

Use the data options to determine data refresh behavior when the file is opened and saved. Some options apply to the entire file, and other options are set per sheet.

Item	Description
Refresh all Axiom functions on open	Determines whether Axiom functions are refreshed when the file is opened. The default setting is **On**. This setting applies to all sheets in the file. Axiom functions such as GetData are non-volatile, and therefore do not refresh automatically—they refresh only on demand, or in response to a change in a dependent cell. If this option is set to **Off**, then the functions will not be refreshed until the user explicitly refreshes the file. For templates/plan files, this option should always be **On** to ensure that driver references are updated when users open plan files. Note that the **Process Plan Files** utility refreshes all functions regardless of this setting. This option does not affect Axiom queries. If you want an Axiom query to refresh when the file is opened, you must enable **Refresh data on file open** for the query. However if an Axiom query is set to refresh on open, this process will also refresh the Axiom functions.
Convert Axiom function results to zero on save	Determines whether Axiom function results are converted to zero when the file is saved. The default setting is **Off**. This setting applies to all sheets in the file. This feature is intended for data security. If the file is not configured to refresh data when the file is opened, then the data queried and saved by one user will still be visible when another user opens the file. This may result in users seeing data outside of their data filter, because the functions will not update until the user explicitly refreshes the file. If this option is set to **On**, then users will see zeros in Axiom functions when they first open the file, and they must refresh the file to see data. This setting is primarily intended for reports. **NOTE:** If this option and **Convert Axiom Query results to zero on save** are both enabled, and the file has Axiom queries that are configured to **Refresh after save data**, then the functions will save as zero but then immediately calculate after the save when the queries are refreshed.

Item	Description
Refresh Forms Run Behavior	Specifies when refresh forms are presented to the user, if the file is configured to display a refresh dialog (using either refresh variables or an Axiom form). This setting applies to all sheets in the file. Available options are: • **Off**: The refresh dialog is disabled and does not display when the file is refreshed. • **OnManualRefreshOnly** (default): The refresh dialog only displays when the file is manually refreshed (by clicking **Refresh**). • **OnOpenOnly**: The refresh dialog only displays 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**: The refresh dialog displays when the file is refreshed on open, and when the file is manually refreshed. The refresh dialog is always disabled 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. For more information, see Setting up refresh dialogs for Axiom files.

Item	Description
Convert Axiom Query results to zero on save	Specifies whether Axiom query data is zeroed when the file is saved. The default setting is **Off**. This setting is configured on a per sheet basis and applies to all Axiom queries on the sheet. This feature is intended for data security within reports. If an Axiom query is *not* set to refresh on open, then the data queried and saved by one user will still be visible when another user opens the file. This may result in users seeing data outside of their data filter, because the query will not update until the user explicitly refreshes the file. If this option is set to **On**, then when the file is saved, all Axiom queries in the sheet are zeroed as follows (depending on the refresh behavior for the query): • If the query is set to rebuild, then all rows in the query are deleted, and the data ranges are collapsed. • If the query is set to update and/or insert, and zero on update is not enabled, then no action occurs. The rows in the data ranges are left as is. • If the query is set to update and/or insert, and zero on update is enabled, then the existing rows are left in the data range, but the data values are zeroed. For more information on which columns are zeroed, see the discussion on zero behavior in Specifying how data is refreshed in an Axiom query. **NOTES:** • The refresh behavior **Refresh during document processing** must be enabled in order for an Axiom query to zero on save. This behavior is enabled by default. • If some Axiom queries use the refresh behavior **Refresh after save data**, then the queries will be zeroed *before* the file is saved. After the file has been saved, the queries will be refreshed. • If an Axiom query has an assigned batch number, that query is not zeroed on save. • If an Axiom query uses **Insert Only** refresh behavior, then the query will be zeroed on save but that zeroed data will never be updated. Either the zero on save option should not be enabled for the sheet, or the query should use a different refresh behavior (such as **Rebuild** or **Insert and Update**).

Item	Description
Enable full AQ query validation mode	Specifies whether all AQ data range filters are sent to the server for parsing as part of the query to the database, or just the first data range filter. This helps determine the data to be returned by the query. • By default this is set to **Off**, which means that only the first data range filter is sent to the server. In the majority of cases, this is sufficient to ensure that the data returned by the data query will be compatible with all data range filters in the report. • If **On**, then all data range filters for the query are sent to the server. This may be necessary in cases where the data range filters specify very different sets of data. Enabling this option will increase the complexity of the data query statement and therefore may slow report performance. It is not recommended to enable this option unless it is necessary for the data query. Contact Axiom Support if you need assistance determining whether this option should be enabled.
Refresh Control Sheet between every AQ	Specifies whether Axiom query settings are read once at the start of the refresh process, or if they are read before running each individual Axiom query. • By default this is set to **Off**, which means that all Axiom query settings are read at the start of the refresh process. In this scenario, Axiom queries cannot be dependent—meaning, the results of one Axiom query cannot affect the settings for another Axiom query. • If **On**, then the Control Sheet is calculated after each Axiom query is run, and the individual query settings are read before each query is run. This option allows for dependent queries. The results of one query can impact the settings of a subsequent query. For example, one of the initial queries might determine the data filter for a subsequent query, or the results might impact whether a subsequent query is active or not. The Control Sheet is always refreshed after all Axiom queries have been run, to update all Control Sheet settings before reapplying features such as freeze panes. **NOTE:** You should only enable this option if you have dependent Axiom queries. Enabling this option can introduce many additional calculation cycles to the refresh process, which may impact performance.

Sheet Filters

You can define sheet filters to filter the data returned by Axiom queries and GetData queries (function or data lookup), for the entire sheet. The **Sheet Filter Type** and **Table or Table Type** fields determine the table or table type to be filtered. The **Filter** defines the specific filter to be applied.

If **Sheet Filter Type** and **Table or Table Type** are defined, but no filter is specified for the sheet, then that row is ignored for the sheet.

For more information, see Defining sheet filters.

Item	Description
Sheet Filter Type	Specifies whether the filter is for a specific table or a table type. Select **By Table** or **By Table Type**.
Table or Table Type	If the filter type is **By Table**, specifies the table to be filtered. If the filter type is **By Table Type**, specifies the table type to be filtered.
Filter	Specifies the filter to be applied to the sheet, for the table or table type selected on the same row. If blank, no filter is applied. Standard filter criteria syntax applies. You can use full Table.Column syntax or column-only syntax. See Filter criteria syntax.

Axiom queries

The remainder of the Control Sheet contains settings for one or more Axiom queries. For details on these settings, see Axiom query settings.

By default, the Control Sheet contains settings for two Axiom queries. If you need more Axiom queries, you can use the Sheet Assistant to add them:

1. In the Axiom Assistant area, select the **Sheet Assistant** tab (if it is not already the active tab).
2. If the Control Sheet is currently the active sheet, move to the sheet where you want to add an Axiom query. The Axiom Query options do not display in the Sheet Assistant if you are currently on the Control Sheet.
3. In the **Axiom Queries** section of the Sheet Assistant, click the **Add Axiom Query** button + to the right of the **Axiom Query** list.

A new Axiom query section is added to the Control Sheet, underneath the last defined section. You can now define the settings for the new query as appropriate, using either the Sheet Assistant or the Control Sheet.

Updating a Control Sheet

From time to time, new features are added to Control Sheets for Axiom files. When this occurs, any files that you have created prior to the introduction of the new feature will not have access to that feature. If you want the file to have access to the new feature, you can update the Control Sheet.

NOTE: If a setting is not present on a Control Sheet, then the default value for that setting is used. You only need to update the Control Sheet if you want to use something other than the default value.

When you update a Control Sheet, all settings from the old Control Sheet are copied to a new version of the Control Sheet. The old Control Sheet is kept in the file and renamed Old_Control_Sheet.

Important: Before you update

Before you update a Control Sheet, you should review the existing file to see if it contains configurations that may not transfer cleanly to a new Control Sheet.

Does the current Control Sheet contain notes or calculations that are outside of the standard Control Sheet settings?

When the Control Sheet is updated, only the content within standard fields will be transferred to the equivalent fields in the new Control Sheet. Anything outside of these fields will not be moved to the new Control Sheet. The non-standard content will remain in the old Control Sheet, and any formula references to the non-standard content will point to the old Control Sheet.

After updating, you should review the old Control Sheet to determine if you need any of the non-standard content. If you do, then we strongly recommend reconfiguring your file to place this content on a "regular" sheet (non-Control Sheet). If you simply move this content to the new Control Sheet, then you will have the same issue the next time you want to update the Control Sheet.

Do the sheets in the file contain formulas that reference the Control Sheet?

If a sheet in the file references a cell in the Control Sheet, these references will be treated as follows:

- If the cell reference is to a standard field on the Control Sheet, then the reference will be maintained and updated as needed. For example, if the reference is to the **Current Dynamic View** setting, and the location of this setting is different between your current Control Sheet and the new Control Sheet, the reference will be updated to point to the appropriate cell on the new Control Sheet.

  NOTE: In a small handful of cases, the formulas cannot be converted to point to the new Control Sheet. This includes references to blank cells in the sheet filter area (references to populated cells are updated).

- If the cell reference is *not* to a standard field on the Control Sheet, then it will not be updated to point to the new Control Sheet. The reference will be maintained, but it will point to the same location on the old Control Sheet.

  In this case, we strongly recommend reconfiguring your file to place this referenced content on a "regular" sheet, and update your formulas accordingly.

If any reference cannot be converted to point to the new Control Sheet, then a warming message will appear after the sheet has been updated, informing you that some formulas now point to the Old_Control_Sheet.

If you are not sure whether your file will update cleanly, then you should make a backup copy of the file before updating the Control Sheet. Even though a copy of the old Control Sheet will remain in the updated file, if any issues occur it may be helpful to view the original configuration. Also, if the updated file needs additional work, you can continue to use the original copy while you adjust the updated file.

Updating the Control Sheet for a single file

You can update Control Sheets on a per file basis. This is the best approach if you want to review the file afterward to make sure that the file updated cleanly.

NOTE: Updating the Control Sheet requires the file to be open with read/write access. Once the Control Sheet is updated, you must be able to save the file.

1. Open the file that you want to update and make sure it is the active file. In the **Axiom Assistant** pane, select **Sheet Assistant** to display the Sheet Assistant task pane.
2. At the top of the Sheet Assistant, click **Update the control sheet**.

   This option is only available if the Sheet Assistant detects that the Control Sheet is not current. If this option does not display, then either the Control Sheet is current and does not need to be updated, or the file is opened read-only.

3. At the confirmation dialog, click **OK** to continue.

   The confirmation dialog reminds you that any non-standard changes to the Control Sheet will be lost when the update occurs. If you have any doubt regarding this file, you should cancel out of the process and make a backup copy of the file before proceeding with the update.

The Control Sheet is updated in the file. You can now use any new features that were added to the Control Sheet, and you can use the Sheet Assistant to modify configuration settings for each sheet.

IMPORTANT: You should review the new Control Sheet and test the file to make sure that it is operating as expected.

Once you have reviewed the file and made any necessary adjustments, you can delete the old Control Sheet. While you can keep the old Control Sheet in the file indefinitely if desired, you may want to eventually delete it in order to avoid confusion when editing or viewing file settings.

Updating the Control Sheet for multiple files

You can use utilities in the standard **QA Diagnostics** task pane or ribbon tab to update the Control Sheet for all files within a designated folder. This functionality is only available to administrators. To do this:

1. Set the currently active file to a file that is not eligible to be processed using the diagnostics tools. For example, click **Reports > New Report** to open a new blank report. Later, you can simply close this new report without saving it.
2. In your **QA Diagnostics** task pane or ribbon tab, in the **Utilities** section, run **Update Control Sheets**.

   You may have renamed this task pane or ribbon tab, or otherwise customized it. If you have not yet created a QA Diagnostics task pane using the standard template, see Using file diagnostics for troubleshooting and optimization. You can also build your own task pane or ribbon tab by using the **Run QA Diagnostics** command.
3. In the **Choose Folder** dialog, navigate to the folder where you want to update the Control Sheets, then click **OK**.
4. At the confirmation prompt, click **Yes** to continue.
   - Axiom cycles through each file in the designated folder.
   - If an out of date Control Sheet is detected, Axiom updates the Control Sheet and saves the file. Otherwise, no action is taken to the file.
5. Once the process is completed, a status message displays the number of documents that were updated. Click **OK**.

After upgrade, the Old_Control_Sheet still remains in the files. You can review any of these files as needed to ensure that the file updated cleanly. After you have performed any necessary verification, you should remove the Old_Control_Sheets from the files. To do this, you can follow the same process

described above, except this time run the **Remove Old Control Sheets** utility. You will be prompted to select a folder, and then the utility will process each file in that folder and remove any Old_Control_ Sheets.

Windows Client Design Considerations

The Axiom Windows Client uses a web-enabled spreadsheet engine to simulate the Microsoft Excel environment. While most spreadsheet and Axiom functionality is available in this web environment, there are some limitations.

This topic details design considerations to keep in mind when designing Axiom files for use in the Windows Client. By default, all Axiom files should follow these design considerations. Axiom files should not use features that are incompatible with the Windows Client unless it is absolutely known that the installation (or the particular file) will only use Excel.

Axiom forms must also follow these design considerations, because the Web Client uses the same web-enabled spreadsheet engine as the Windows Client. Although form users do not see the spreadsheet source file, it is opened on the server using the spreadsheet engine.

Functions

Most Microsoft Excel functions are supported for use in the Windows Client. Some functions are not supported, or may have behavior differences.

All Axiom functions are supported for use in the Windows Client.

Unsupported functions

The following functions are not supported:

- ASC
- BAHTTEXT
- CONCAT
- CALL
- DBCS
- EUROCONVERT
- FIELDVALUE
- FILTER
- FORMULATEXT
- GETPIVOTDATA

- IFS
- JIS
- LET
- MAXIFS
- MINIFS
- PHONETIC
- RANDARRAY
- REGISTER.ID
- RTD
- SEQUENCE
- SORT
- SORTBY
- SQL.REQUEST
- SWITCH
- TEXTJOIN
- UNIQUE
- XLOOKUP
- XMATCH
- Any cube-related function (CUBEKPIMEMBER, CUBEVALUE, etc.)

Function behavior differences

Note the following function behavior differences between the Windows Client and Excel:

Function	Behavior Differences
CEILING	The behavior of the function CEILING is the old Excel behavior. In older versions of Excel, if the number and significance have different signs, CEILING returns the #NUM! error value. In current versions of Excel, if the number and significance have different signs, the number is rounded up or down toward zero.
CELL	The Windows Client requires the second parameter of CELL to specify the cell that you want to return information about, whereas Excel does not require this parameter. If the second parameter is omitted, then the Windows Client will return an error. Also, when using CELL to return the file name, the Windows Client returns the location in the Axiom file system, whereas the Excel returns the location of the temporary file on the client workstation.
HYPERLINK	In the Windows Client, HYPERLINK cannot be used to link within the current file or to another file. However, it can be used to link to a web page URL.

Function	Behavior Differences
INDIRECT	Use of INDIRECT to other sheets can cause issues in the Windows Client if the sheet name has a space. To work around this issue, do not use spaces in the sheet names (an underscore is a good alternative).
MATCH	When the values -1 and 1 are used in the third parameter of the MATCH function, the Excel Client and the Windows Client may return different values if the data is not sorted according to the Excel MATCH documentation.

Spreadsheet features

Most Microsoft Excel spreadsheet features are supported in the Windows Client. Some features are not supported, and some may behave differently in the Windows Client environment. If Axiom files are to be used in the Windows Client, make sure to test in that environment to help identify any feature that may not be working as expected.

Unsupported spreadsheet features

Some Microsoft Excel features that were *new* in Excel 2007 and up are *not* supported—for example, sparklines, slicers, and improved charting. However, this does not mean that your spreadsheets must be 100% compatible with Excel 2003 in order to work in the Windows Client. For example, the expanded row and column limits of up to 1 million rows and columns out to XFD are fully supported.

Other spreadsheet limitations include:

- The new dynamic array features in Office 365 are not supported in the Windows Client (or in previous versions of Excel), and may cause errors if the file is used and saved outside of the Office 365 environment. Please see Office 365 and dynamic arrays for more information.
- Cell comments and form controls (buttons, check boxes, list boxes, etc.) are not supported in the Windows Client, and will be removed if the file is saved in the Windows Client.
- Table references in formulas—such as [Sales]—are converted to `#REF!`. Excel Tables are not supported in the Windows Client.
- Reading and writing shapes is limited. Complex shapes will be lost when a file is saved in the Windows Client, and other shape properties may be lost. It is not recommended to use shapes.
- Certain image formats are not supported and will be removed when the file is saved in the Windows Client: GIF, EMF, WMF.
- Visual Basic (VBA) is not supported in the Windows Client. When XLSM files are opened in the Windows Client, any VBA in the file is preserved for later use in the Excel Client, but it is not available for use in the Windows Client.
- Cell indenting is not supported in the Windows Client. Indented text is displayed, but without the indent.

- Superscript text is not supported in the Windows Client. The text is displayed, but not with superscript formatting.
- The "shrink to fit" cell formatting option is not supported in the Windows Client.
- Pivot tables are not supported in the Windows Client, and will be "flattened" when the file is saved in the Windows Client.
- Sheet protection from Excel 2013 or up, if manually applied to a sheet, will prevent the file from being opened in the Windows client. Sheet protection should always be applied via the Control Sheet.
- Copying and pasting content into the Windows Client is limited to 65536 rows. When pasting content copied from Excel, any content that exceeds this limit will not be copied.

TIP: If something is not working in the Windows Client as expected, try saving a copy of the file within Excel as an XLS file. Review the warnings in the compatibility checker to see if you may be using a newer Excel feature that is not supported in the Windows Client.

Spreadsheet feature behavior differences

Some spreadsheet features, while supported, may work slightly differently in the Windows Client versus the Excel Client:

- Data Validation drop-down lists do not work in the Windows Client if they are located within frozen panes.
- Data Validation sheet references will not work in the Windows Client, unless they are wrapped in an Indirect function. For example: `Indirect("Info!A1:A10")`.
- The sheet protection settings applied by Axiom work slightly differently between the Excel Client and the Windows Client. For more information, see Configuring worksheet protection for Axiom files.
- The Excel feature Insert Hyperlink is supported by the Windows Client, but not to link to another file. It can be used to link to a web page, or to link to a place within the current document.

Axiom file features

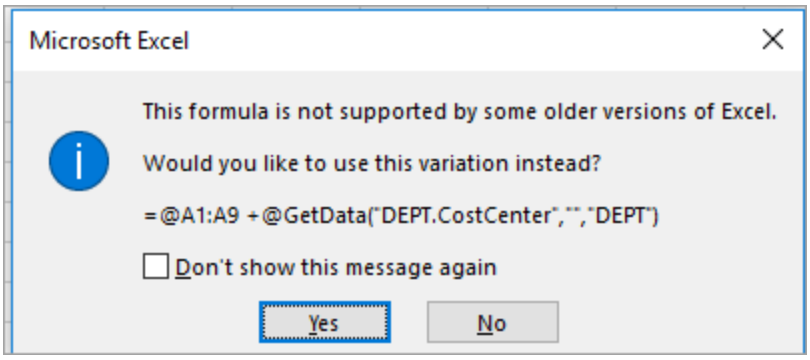
All Axiom file features are available in both the Windows Client and the Excel Client, such as Axiom queries, Axiom functions, save-to-database, calc methods, and file formatting / display options such as GoTo bookmarks, view controls, and snapshots.

Some non-Axiom features that leverage Excel functionality are not available or limited in the Windows Client, such as the ability to save files to Sharepoint.

Office 365 and dynamic arrays

Starting in March 2020, Office 365 monthly releases included a new feature named "dynamic arrays," which introduces substantial changes to the Excel calculation engine. This new feature is not compatible with earlier versions of Excel or the Axiom Windows Client. At this time, any functionality relating to dynamic arrays should not be used in Axiom files, as it may introduce errors when the files are opened or saved in any environment other than Office 365.

Use caution when developing Axiom spreadsheet files in Office 365 to ensure that you are not inadvertently introducing incompatible formulas. After entering a formula into Office 365, you may see a message informing you that the formula is incompatible and suggesting a compatible version. ***Always choose Yes when you see this prompt.*** Additionally, do NOT select the option **Don't show this message again**, as this will prevent you from being prompted about incompatible formulas in the future.



If you accidentally select **Don't show this message again**, you can restore the warning in Excel by going to **File > Options > Formulas** and selecting **Suggest formula variations that are supported by older versions of Excel**.

Reference

Filter criteria syntax

Several areas of Axiom use criteria statements to define a set of data. The syntax for these criteria statement is as follows:

```
Table.Column='Value'
```

- *Table* is the name of the database table.
- *Column* is the name of the column in the database table.
- *Value* is the value in the column.

If the column is String, Date, or DateTime, the value must be placed in single quotation marks as shown above. If the column is Numeric, Integer (all types), Identity, or Boolean, then the quotation marks are omitted.

For example:

- To filter data by regions, the filter criteria statement might be: `DEPT.Region='North'`. This would limit data to only those departments that are assigned to region North in the Region column.
- To filter data by a single department, the filter criteria statement might be: `DEPT.Dept=100`. This would limit data to only department 100.

If the table portion of the syntax is omitted, then the table is assumed based on the current context. For example, if the filter is used in an Axiom query, then the primary table for the Axiom query is assumed. If the current context supports *column-only syntax*, and the specified column is a validated key column, then the lookup table is assumed.

Operators

The criteria statement operator can be one of the following: =, >,<,<>,<=,>=. Greater than or less than statements can only be used with numeric values. For example:

```
ACCT.Acct>1000
```

SQL IN and LIKE syntax can also be used. For example:

```
DEPT.Region IN ('North','South')
```

Compound criteria statements

You can use `AND` and `OR` to combine multiple criteria statements. If you are creating long compound criteria statements with multiple ANDs or ORs, you can use parentheses to group statements and eliminate ambiguity. For example:

```
(DEPT.Region='North' OR DEPT.Region='South') AND (ACCT.Acct=100 OR
ACCT.Acct=200)
```

NOTES:

- When filtering on multiple values in the same column, you must use OR to join the statements, not AND. In the example above, if the statement was instead `DEPT.Region='North' AND DEPT.Region='South'`, that statement would return no data because no single department belongs to both the North and South regions. When you use OR, the statement will return departments that belong to either the North or the South regions.
- Alternatively, you can use the SQL IN syntax to create a compound statement for values in the same column. For example, the statement `DEPT.Region='North' OR DEPT.Region='South'` can also be written as `DEPT.Region IN ('North','South')`. The Filter Wizard uses IN syntax by default.

Using criteria statements in functions

If you are using a criteria statement in a function, such as GetData, you must place the entire criteria statement in double quotation marks. For example:

```
=GetData("Bud1","DEPT.Region='North'","GL1")
```

You can also place the criteria statement in a cell and then use a cell reference in the function. In this case, you do not need to use double quotation marks in the function, unless you are concatenating text and cell reference contents within the function.

Referencing blank values in filters

If a string column contains a blank value, you may want to create a filter that includes or excludes records with these blank values. For SQL Server, the blank value is stored as an empty string. This empty string is indicated with empty quotation marks in the filter. For example: `ACCT.CMAssign=''` or `ACCT.CMAssign<>''`

If you use the Filter Wizard to construct the filter, it will automatically use the appropriate syntax.

Referencing values with apostrophes in filters

If a string column contains a value with an apostrophe (such as O'Connor), then that apostrophe must be escaped with another apostrophe so that it is not read as the closing apostrophe for the filter criteria statement. For example:

`Dept.VP='O'Connor'`

Invalid. This construction does not work because Axiom reads it as Dept.VP='O' and then does not know what to do with the rest of the text.

`Dept.VP='O''Connor'`

Valid. The extra apostrophe tells Axiom that the apostrophe is part of the string value and is not the closing apostrophe.

NOTE: This syntax must use two apostrophe characters in sequence and *not* a double quotation mark. If you create the filter using the Filter Wizard, Axiom will construct the appropriate syntax for you.

Referencing Date or DateTime values in filters

If your locale uses a date format where the first value is the day, filters using that date or date-time value will not process correctly. Instead, the date or date-time value must be in standard format. Standard format is `YYYY-MM-DDTHH:MM:SS` for DateTime and `YYYY-MM-DD` for Date.

If you use the Filter Wizard to construct the filter, it will automatically convert the date or date-time value to the appropriate syntax.

Using formulas with Axiom feature tags

Axiom supports a number of features that depend on using reserved tags within a spreadsheet. For example, save-to-database operations, action codes, sheet views, and data sources for Axiom forms all use these feature tags.

Generally speaking, feature tags have the following components:

- A "primary tag" that enables use of the feature within the sheet. This tag defines the control row and control column for the feature, and may also contain additional parameters that control options for the feature.
- Various column tags to be placed in the control row, to define the operation of the feature.
- Various row tags to be placed in the control column, to define the operation of the feature.

For example, `[Save2DB;TableName]` is the primary tag for the Save Type 1 save-to-database feature. The "Save2DB" part of the tag identifies the feature, and the rest of the tag defines parameters for the feature (there are other optional parameters not shown here). The column names that determine the destination columns for the save are the column tags, and the `[Save]` tag that flags rows to be saved are the row tags.

Using formulas with primary tags

You can use formulas to create primary tags instead of "hard-coding" the tag within the cell. However, the initial bracket and the identifying text must be present as a whole in the formula. You cannot concatenate this part of the tag or reference another cell that contains this tag. For example:

Valid `="[Save2DB;"&C13&"]"`

This is valid because the initial bracket and the identifying text “Save2DB” are present as a whole in the formula. Axiom looks for the text "[Save2DB" in order to determine if the feature exists in the sheet.

Invalid `="["&"Save2DB;" &C13&"]"`

This is invalid because the bracket and identifying text are the result of a concatenation.

Invalid `=P4&C13`

This is invalid because the bracket and identifying text are not contained within the formula. This formula might resolve to what looks like a valid tag within the spreadsheet (depending on what is in P4 and C13), but it will not be found by Axiom.

Once Axiom has identified that a cell contains a primary tag, the formula result is evaluated as normal to determine if the feature is actually applied. For example, if the formula contains "[Save2DB" then Axiom recognizes it as a candidate for processing. But if the formula uses an IF statement and therefore the actual result of the formula in the file is a blank cell or other text such as "No Save," then that cell will be skipped for processing.

If the primary tag takes parameters, the parameters can be created by concatenation or by cell reference, without restrictions. All of the following examples are valid:

```
[Save2DB;Plan2020]
=”[Save2DB;”&C13&”]”
=”[Save2DB;”&”Plan”&”2020]”
```

Using formulas with column and row tags

Once a primary tag has been found in the sheet and identified for processing, its associated column and row tags are fully evaluated. You can use any kind of formula construction to create these tags.

For example, the row tag `[Save]` can be created by concatenation or by cell reference, without restrictions. All of the following examples are valid:

```
[Save]
=”[“&”Save”&”]”
=C24
```

Double-click behavior

Axiom supports several different features that can be initiated by double-clicking a cell in a plan file or a report. The following priority order determines what happens when a particular cell is double-clicked. If the first item in the list does not apply to the cell, the second item is evaluated, and so on.

1. If the cell contains an Axiom function that triggers on double-click (GetDocument, GetCalcMethod, etc.), then that action is performed.
2. If the cell is part of a validated calc method that has calc method variables, the associated calc method form is opened.
3. If custom drilling is configured for the file, and double-click has been enabled for the custom drill, the custom drill is performed.
4. If the cell is eligible for drill-down or drill-through drilling, the **Select Drill Option** dialog is opened.

If none of the above apply, the normal Microsoft Excel spreadsheet behavior occurs. In most cases the behavior is to place your cursor in the cell for editing.

NOTE: Items 2 and 4 require **Axiom Double-Click** to be enabled for the sheet (by using the Sheet Assistant or the Control Sheet).

Index

C

D

F

G

H

I

L

M

N

O

T

U

V

W

Z