What's New

Axiom Software

Version 2020.1

Contents

Introduction

Kaufman Hall is pleased to announce the release of **Axiom Software Version 2020.1**. This release features enhancements to various areas of the software, such as:

- **Data Grid and Fixed Report components** in Axiom forms were enhanced to meet a broader range of use cases:
  - Ability to edit data in a data grid and save changed data to the database
  - Option to perform calculations in both components based on pre-aggregated data
  - Define conditional calculations in both components so that rows can use different calculations based on specified conditions
- **Plan file processes** now provide flexible step rejection behavior, so that the step owner can select the step to return to when rejecting a plan file
- **Calc method variables** now support additional variable types and the ability to dynamically change the variable configuration based on information in the file

This *What's New* document provides information on all new features and enhancements in this release. Reviewing this document should give you a basic understanding of how these new features work, and what benefits they may provide to your organization. For full details on any new feature, please see the Axiom Software Help files or the PDF guides.

> **IMPORTANT:** Before upgrading to version 2020.1, make sure you have reviewed the separate *Release Notes* document to understand any important technical changes and upgrade considerations in this release.

Axiom forms

This section details the new features and enhancements made to Axiom forms.

Save data from Data Grid components

You can now edit values in a Data Grid component and then save changed data back to the database. This feature expands the available options in Axiom forms to allow users to review, edit, and save data.

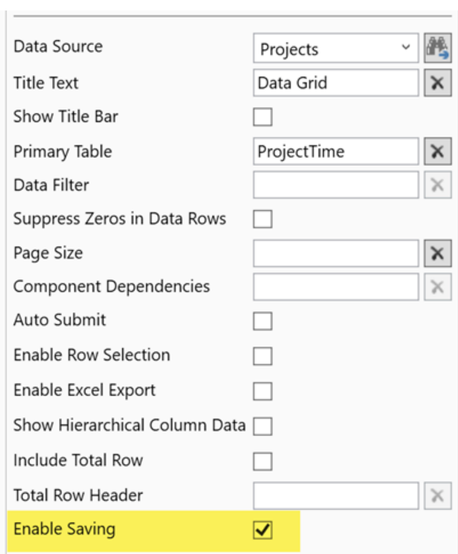
This feature works as follows:

- You can configure one or more data grid columns as editable, so that users can edit the value in the grid by typing into the cell.
- You can configure calculated columns to update live, so that if the calculation references an editable column, the value in the calculated column updates in response to the user's input.
- You can enable saving data for the grid, and specify a target table and columns for the save. When a save-to-database is triggered in the form, any grid rows with edited data will be saved to the database.

Configuring the grid for editing and saving data

The component properties and the DataGridColumns data source contain new fields to configure the ability to edit and save data in a Data Grid component.

1. In the component properties, select **Enable Saving**.



2. In the DataGridColumns data source, for each column where you want the user to be able to edit values, enter `True` into the `[IsEditable]` column.

	A	B	C	D	E	F
5						
6			[DataGridColumns;Projects]	[ColumnName]	[IsVisible]	[IsEditable]
7			[Column]	ProjectTime.ProjectTime		
8			[Column]	ProjectTime.Project	TRUE	
9			[Column]	ProjectTime.StartDate	TRUE	
10			[Column]	ProjectTime.BilledHours	TRUE	TRUE
11			[Column]	ProjectTime.Rate	TRUE	TRUE
12			[CalculatedColumn]	ProjectTotal	TRUE	
13			[Column]	ProjectTime.Consultant	TRUE	

NOTE: If you use the right-click wizard to create a new DataGridColumns data source, the `[IsEditable]` tag and the other new data source tags will be present by default. If you want to modify an existing DataGridColumns data source to enable editing and saving, then you need to add the new tags manually. Remember that these tags can be in any order.

3. In the DataGridColumns data source, for each calculated column that you want to update in response to user edits, enter `True` into the `[IsLiveUpdate]` column.

	A	B	C	D	E	F	I	K	L	T	V
5											
6			[DataGridColumns;Projects]	[ColumnName]	[IsVisible]	[IsEditable]	[IsSumBy]	[Header]	[Icon]	[Calculation]	[IsLiveUpdate]
7			[Column]	ProjectTime.ProjectTime			TRUE				
8			[Column]	ProjectTime.Project	TRUE			Project	OpenDoc		
9			[Column]	ProjectTime.StartDate	TRUE			Start			
10			[Column]	ProjectTime.BilledHours	TRUE	TRUE		Hours			
11			[Column]	ProjectTime.Rate	TRUE	TRUE		Rate			
12			[CalculatedColumn]	ProjectTotal	TRUE			Total		ProjectTime.BilledHours*ProjectTime.Rate	TRUE
13			[Column]	ProjectTime.Consultant	TRUE			Consultant			

4. In the DataGridColumns data source, configure the `[Save;TABLENAME]` column as follows:
   - Replace TABLENAME with the name of the target table for the save-to-database. For example, change the column tag to `[Save;BGT2021]` if you want to save edited grid data to that table.
   - For each column in the grid to be saved, enter the name of the target column in the target table. All key columns must be included in the save, as well as any other columns for which you want to save changed data.

	A	B	C	D	E	F	I	K	L	T	V	W
5												
6			[DataGridColumns;Projects]	[ColumnName]	[IsVisible]	[IsEditable]	[IsSumBy]	[Header]	[Icon]	[Calculation]	[IsLiveUpdate]	[Save;ProjectTime]
7			[Column]	ProjectTime.ProjectTime			TRUE					ProjectTime
8			[Column]	ProjectTime.Project	TRUE			Project	OpenDoc			
9			[Column]	ProjectTime.StartDate	TRUE			Start				
10			[Column]	ProjectTime.BilledHours	TRUE	TRUE		Hours				BilledHours
11			[Column]	ProjectTime.Rate	TRUE	TRUE		Rate				Rate
12			[CalculatedColumn]	ProjectTotal	TRUE			Total		ProjectTime.BilledHours*ProjectTime.Rate	TRUE	ProjectTotal
13			[Column]	ProjectTime.Consultant	TRUE			Consultant				

To trigger the save-to-database in the form, you can use either of the following:

- An interactive component with **Save on Submit** enabled. Typically this is a Button component, however, the save can be triggered by any form component that supports the Save on Submit property.
- The save button in the gray Task Bar, if this button is enabled for the form.

Example

In the following example, the BilledHours and Rate columns have been configured as editable. A calculated column is used to show the results of BilledHours*Rate, and this column is configured for live updates, so that it responds to edits in either of the source columns. Lastly, the save property has been configured to save edits and the calculated value back to the ProjectTime table.

	A	B	C	D	E	F	I	K	L	T	V	W
5												
6			[DataGridColumns;Projects]	[ColumnName]	[IsVisible]	[IsEditable]	[IsSumBy]	[Header]	[Icon]	[Calculation]	[IsLiveUpdate]	[Save;ProjectTime]
7			[Column]	ProjectTime.ProjectTime			TRUE					ProjectTime
8			[Column]	ProjectTime.Project	TRUE			Project	OpenDoc			
9			[Column]	ProjectTime.StartDate	TRUE			Start				
10			[Column]	ProjectTime.BilledHours	TRUE	TRUE		Hours				BilledHours
11			[Column]	ProjectTime.Rate	TRUE	TRUE		Rate				Rate
12			[CalculatedColumn]	ProjectTotal	TRUE			Total		ProjectTime.BilledHours*ProjectTime.Rate	TRUE	ProjectTotal
13			[Column]	ProjectTime.Consultant	TRUE			Consultant				

Example DataGridColumns data source configured to allow edits and save data

When the form is rendered, the editable column values display in editable cells:

Project Tracker

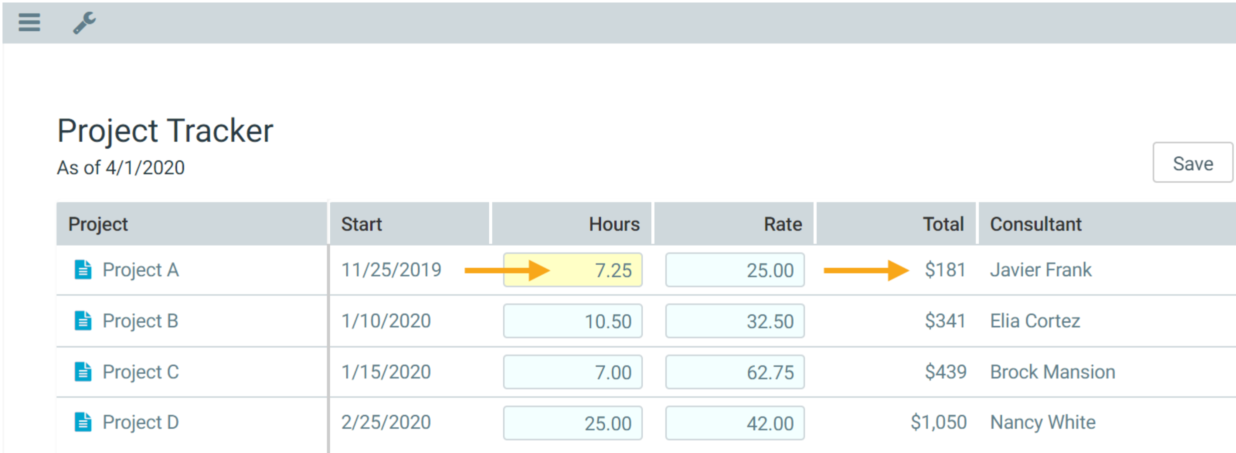
As of 4/1/2020

Save

Project	Start	Hours	Rate	Total	Consultant
Project A	11/25/2019	5.00	25.00	$125	Javier Frank
Project B	1/10/2020	10.50	32.50	$341	Elia Cortez
Project C	1/15/2020	7.00	62.75	$439	Brock Mansion
Project D	2/25/2020	25.00	42.00	$1,050	Nancy White

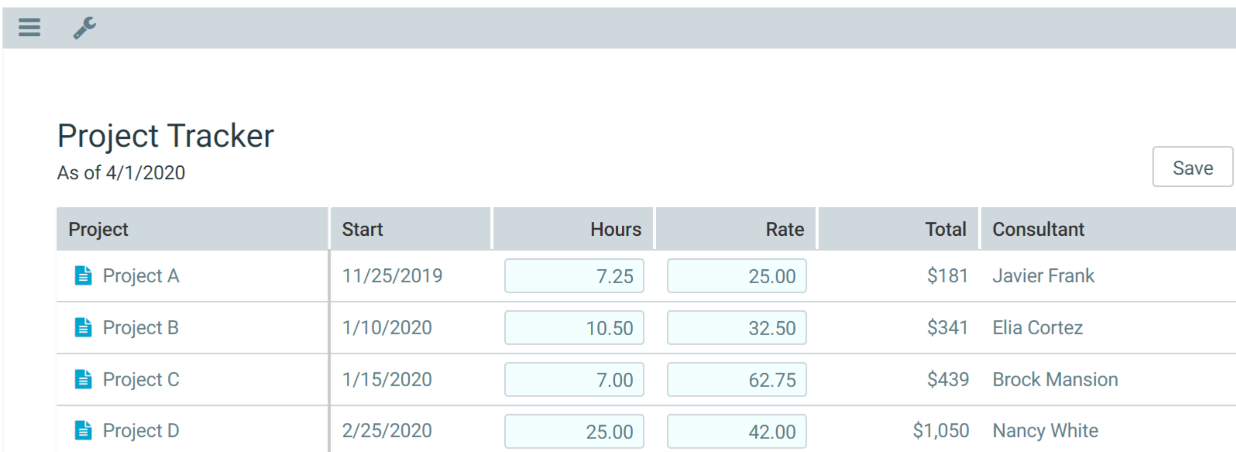
Example form with editable columns

When an edit is made to a value, the background color of the cell changes to yellow, to indicate that the grid contains an unsaved change. In this example, the hours for Project A were changed to 7.25. Because the calculated column Total is configured for live updates, it has updated to show the new total based on the edited value.



Example form after making an edit

Once the save button has been used to save changes to the database, the grid updates to display the latest data from the database. The background color of the edited cell reverts back to blue, because now it is showing the queried value from the database.



Example form after saving edits

Overview of limitations and requirements

The ability to edit and save data from a Data Grid component is intended to meet a fairly narrow use case. You can query data from the database, edit values in designated columns, and then save the edited values and/or save calculated values which reference the edited values. Generally speaking, it is not possible to provide additional edit controls like drop-down lists of values, and it is not supported to create new records within the grid.

Additionally, the edit and save functionality is only supported for relatively simple grid configurations. For example, paged data, hierarchical groupings, and selecting rows are not supported. The total row can be enabled, but does not update in response to user edits.

For additional information on limitations, see the following topic in Axiom Software Help: *Editing and saving data using a Data Grid* (AX1124).

How the data grid save works

When the Data Grid component is rendered in the form, any columns that are configured as editable display with their values in editable cells. For consistency, these cells use the same formatting as other editable form components, such as the Text Box component or editable cells in a Formatted Grid component. The editable cells are outlined and have a light blue background.

To edit a cell value, the user can click into the cell and then type. When the user clicks or tabs out of the cell after editing the value, the cell now displays with a light yellow background. This is intended as a signal to the user that the value has been changed, but has not yet been saved.

When the user triggers a save-to-database in the form, any edited rows in the grid are saved to the database, using the target table and columns as configured in the DataGridColumns data source. Unlike standard form saves, the data grid save is performed at the start of the form update cycle, before any other form processing begins. If the form contains an Axiom query that references the target table of the data grid save, the updated data is available to the query.

The data grid is refreshed at the end of the form update cycle, so that it displays the most current data from the database. Any cells in the grid that were previously formatted as changed are now restored to their original formatting.

For additional information on design considerations, see the following topic in Axiom Software Help: *Editing and saving data using a Data Grid* (AX1124).

Use pre-aggregation calculations in Data Grid and Fixed Report components

You can now use pre-aggregation calculations in both the Data Grid component and the Fixed Report component. This flexibility allows the components to accommodate a broader range of use cases.

By default, calculations are performed post-aggregation, which means the calculation is applied to the data rows as they display in the component. In other words, the calculation is applied at the "sum by" level of the rows. Now, you can optionally apply the calculation to the raw data records returned by the query, before data is aggregated based on the sum by level. This is known as a pre-aggregation calculation.

Configuring a calculation as pre-aggregation

A new column tag of `[IsPreAggregationCalculation]` is available in both the DataGridColumns data source and the FixedReportColumns data source. To enable a calculation to be performed pre-aggregation, enter True in this column. False is assumed if the column is blank or omitted, so all existing calculations will continue to be post-aggregation unless you configure them otherwise.

NOTE: If you use the right-click wizard to create a new DataGridColumns or FixedReportColumns data source, the `[IsPreAggregationCalculation]` tag will be present by default. If you want to modify an existing data source to enable pre-aggregation, then you need to add the new tag manually. Remember that tags can be in any order.

If you want a calculation to be performed pre-aggregation, some limitations apply:

- Pre-aggregation calculations can only use database column names, and those database columns must be present on the primary table or a lookup table.
- Pre-aggregation calculations cannot reference other calculated columns defined in the data source, and cannot reference named columns in the data source.

Total and subtotal rows in data grids and fixed reports now behave differently depending on whether the calculation is pre-aggregation or post-aggregation:

- If the calculation is post-aggregation, then the calculation is applied to the values in the total or subtotal row. This is the same behavior as in previous versions.
- If the calculation is pre-aggregation, then the total or subtotal row displays the sum of values in the column.

Example of pre-aggregation versus post-aggregation calculations

Imagine that you have the following rows of data in the database, and the sum by level of the grid or report rows is Dept.Region. These rows will be aggregated (summed) together to result in one Region West row in the component.

Dept	Region	Value1	Value2
100	West	5	1.25
200	West	10	2

If a calculation is defined of `Dept.Value1 * Dept.Value2`, and the calculation is applied post-aggregation, then the value for Region West is calculated as follows:

- First the two Region West rows are aggregated to result in 5+10=15 for Value1 and 1.25+2=3.25 for Value2.

Region	Value1	Value2
West	15	3.25

- Then the calculation of Value1 * Value2 is applied to the aggregated data, resulting in 15 * 3.25=**48.75**.

Region	Value1	Value2	Calculation
West	15	3.25	48.75

If the same calculation is applied pre-aggregation, then the value for Region West is calculated as follows:

- First the calculation of Value1 * Value2 is applied to each pre-aggregated row of the data, resulting in values of 5 * 1.25=6.25 and 10 * 2=20 respectively.

Dept	Region	Value1	Value2	Calculation
100	West	5	1.25	6.25
200	West	10	2	20

- Then the two Region West rows are aggregated to result in the calculated value 6.25+20=**26.25**.

Region	Value1	Value2	Calculation
West	15	3.25	26.25

Use conditional calculations in Data Grid and Fixed Report components

When setting up calculations in a Data Grid or Fixed Report component, you may want different rows to use different calculations. For example, when showing employee data, you might want to use different calculations for salaried employees versus hourly employees. You can now do this in both components by using the new conditional calculations feature.

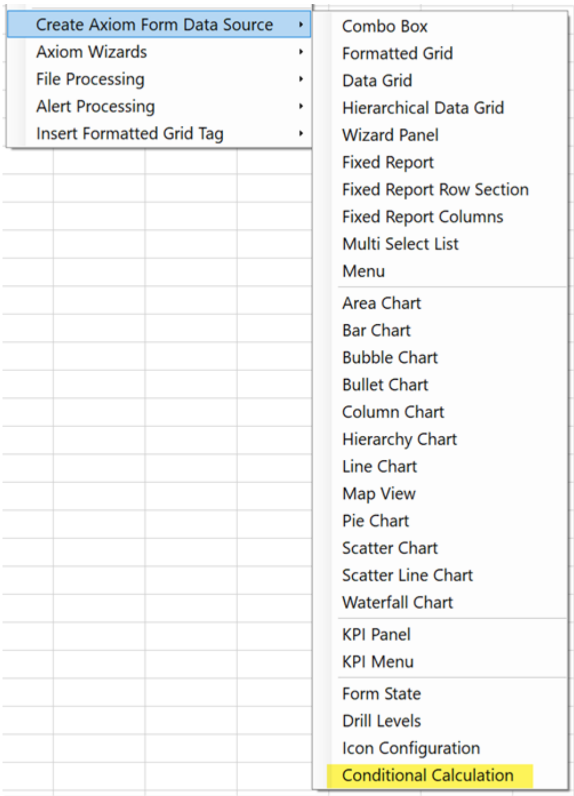
To set up conditional calculations for either a Data Grid or Fixed Report component, you do the following:

- Create a ConditonalCalculation data source to define the conditions. Each row of this data source defines a condition and its associated calculation.
- In the DataGridColumns or FixedReportColumns data source, enter the name of the ConditionalCalculation data source into the `[Calculation]` property.

When the component is rendered in the form, each row of the grid or report is evaluated against the conditions in the ConditionalCalculation data source to determine which calculation to use.

Creating the ConditionalCalculation data source

To create the ConditionalCalculation data source, you can use the right-click wizard: **Create Axiom Form Data Source > Conditional Calculation**.



New data source option in right-click wizard

Each row of the data source defines the following:

- A `[Condition]` to evaluate against each row in the grid or report, in a format such as `Acct.Acct=4000`. The usual filter syntax and operators are supported to define the condition.
- A `[Calculation]` to apply to rows that match the condition. This property can use the same syntax and operators as the `[Calculation]` field in the DataGridColumns or FixedReportColumns data source.

	C	D	E	F
45				
46		[ConditionalCalculation;VariableCalc]	[Condition]	[Calculation]
47		[Item]	acct.acct=5300	GL2019.TOT*.025
48		[Item]	acct.acct=5400	GL2019.TOT*.04
49		[Item]		GL2019.TOT

Example ConditionalCalculation data source

To use this data source in the data grid, you would create a `[CalculatedColumn]` row in the DataGridColumns data source, and then enter the name of the ConditionalCalculation data source name into the `[Calculation]` column. In this example, you would enter `VariableCalc`.

Each row in the component is tested against the conditions in the data source, in the order that the items are listed in the data source. If a row matches the condition, then the corresponding calculation is used for that row. No further conditions are evaluated for that row.

The data source can contain a row with a blank condition, to serve as a catch-all for any rows in the component that do not match any of the other defined conditions. If you want to use a blank row, that row must be the final row in the ConditionalCalculation data source.

In this example, rows matching account 5300 use the first defined calculation, and rows matching account 5400 use the second calculation. All other rows use the final calculation with a blank condition.

For more information and additional design considerations, see the following topics in Axiom Software Help:

- *Using conditional calculations in Data Grid components* (AX1041)
- *Using conditional calculations in Fixed Report components* (AX1084)

Conditional calculation example

Imagine that you want to display an employee roster in a Data Grid component, with a calculation of their projected salary for the next year. The data and calculations that you want to apply may vary depending on whether the employee is hourly or salaried.

This example uses conditional calculations for two purposes:

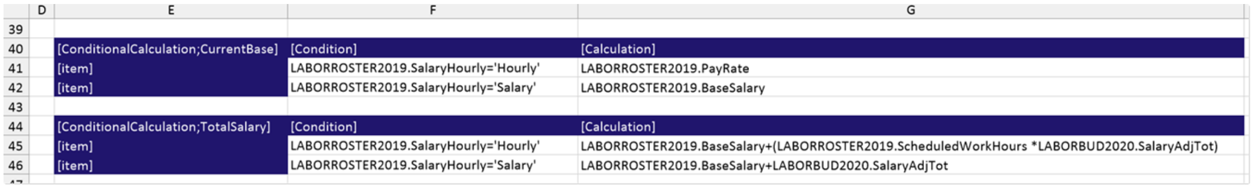
- To display the current base pay for each employee, you want to return the value from different columns depending on whether the employee is hourly or salaried. This calculation references the CurrentBase data source.
- To display the calculated current salary for each employee, the calculation needs to be made differently depending on whether the employee is hourly or salaried. This calculation references the Total Salary data source.

	D	E	F	G	H	K	L	M	R	AA
27										
28		[DataGridColumns;Roster]	[ColumnName]	[IsVisible]	[DisplayFormat]	[IsSumBy]	[Width]	[Header]	[Total]	[Calculation]
29		[Column]	LABORROSTER2019.Labor_Position	TRUE	{LABORROSTER2019.Labor_Position} - {LAE	TRUE	300	Position ID		
30		[Column]	LABORROSTER2019.Labor_Position.Description	FALSE		TRUE	200	Description		
31		[Column]	LABORROSTER2019.Labor_Employee	TRUE	{LABORROSTER2019.Labor_Employee.Emp	TRUE	200	Employee		
32		[Column]	LABORROSTER2019.SalaryHourly	TRUE		FALSE	125	Class		
33		[CalculatedColumn]	CurrentBase	TRUE		FALSE	150	Current Base Pay	FALSE	CurrentBase
34		[Column]	LABORBUD2020.SalaryAdjTot	TRUE		FALSE	150	Salary Adjustment	FALSE	
35		[Column]	LABORROSTER2019.BaseSalary;BaseSal	TRUE		FALSE	180	Calculated Current Salary	TRUE	
36		[CalculatedColumn]	TotalSalaryCalc	TRUE		FALSE	150	Proposed Salary	TRUE	TotalSalary

Example calculated columns referencing ConditionalCalculation data sources

In the ConditionalCalculation data sources, there are different conditions defined to determine whether the employee is hourly or salaried, and then apply the appropriate calculation.

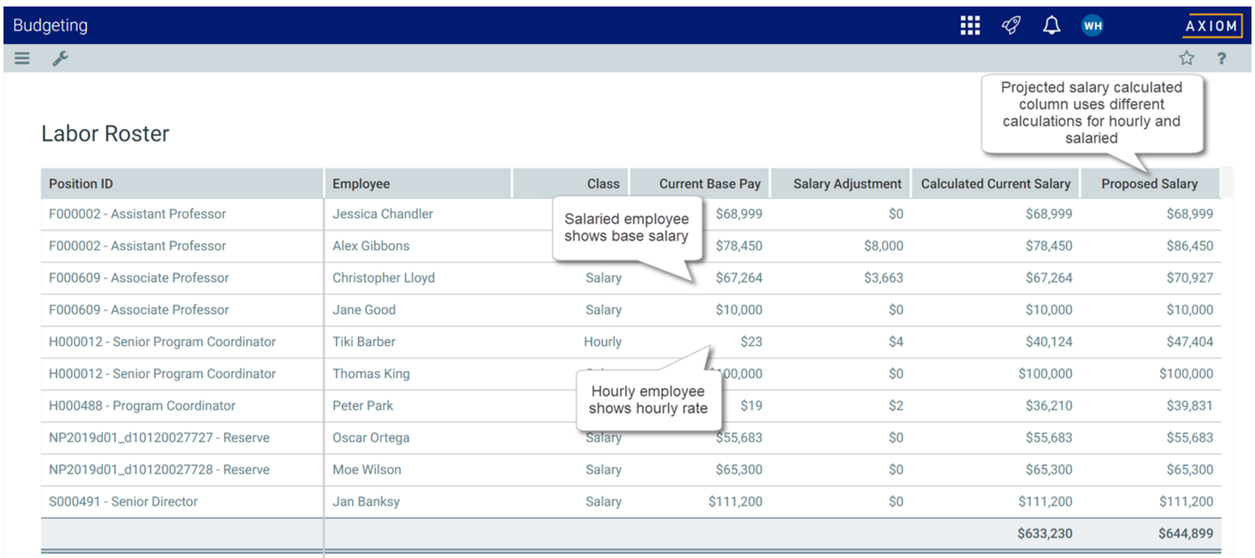
- The CurrentBase data source uses the calculation field to display values from different table columns for each employee—the pay rate for hourly employees, and the base salary for salaried employees.
- The TotalSalary data source uses the calculation field to apply a different calculation for each employee. The calculation for hourly employees incorporates their scheduled work hours to calculate the projected salary, whereas the calculation for salaried employees can simply add the proposed adjustment to their base salary.



	D	E	F	G
39				
40		[ConditionalCalculation;CurrentBase]	[Condition]	[Calculation]
41		[item]	LABORROSTER2019.SalaryHourly='Hourly'	LABORROSTER2019.PayRate
42		[item]	LABORROSTER2019.SalaryHourly='Salary'	LABORROSTER2019.BaseSalary
43				
44		[ConditionalCalculation;TotalSalary]	[Condition]	[Calculation]
45		[item]	LABORROSTER2019.SalaryHourly='Hourly'	LABORROSTER2019.BaseSalary+(LABORROSTER2019.ScheduledWorkHours *LABORBUD2020.SalaryAdjTot)
46		[item]	LABORROSTER2019.SalaryHourly='Salary'	LABORROSTER2019.BaseSalary+LABORBUD2020.SalaryAdjTot

Example ConditionalCalculation data sources to define different calculations for hourly versus salaried employees

In the rendered data grid, the rows associated with salaried employees and hourly employees use the corresponding calculations from the CurrentBase and TotalSalary data sources.



Position ID	Employee	Class	Current Base Pay	Salary Adjustment	Calculated Current Salary	Proposed Salary
F000002 - Assistant Professor	Jessica Chandler		$68,999	$0	$68,999	$68,999
F000002 - Assistant Professor	Alex Gibbons		$78,450	$8,000	$78,450	$86,450
F000609 - Associate Professor	Christopher Lloyd	Salary	$67,264	$3,663	$67,264	$70,927
F000609 - Associate Professor	Jane Good	Salary	$10,000	$0	$10,000	$10,000
H000012 - Senior Program Coordinator	Tiki Barber	Hourly	$23	$4	$40,124	$47,404
H000012 - Senior Program Coordinator	Thomas King		00,000	$0	$100,000	$100,000
H000488 - Program Coordinator	Peter Park		$19	$2	$36,210	$39,831
NP2019d01_d10120027727 - Reserve	Oscar Ortega	Salary	$55,683	$0	$55,683	$55,683
NP2019d01_d10120027728 - Reserve	Moe Wilson	Salary	$65,300	$0	$65,300	$65,300
S000491 - Senior Director	Jan Banksy	Salary	$111,200	$0	$111,200	$111,200
					$633,230	$644,899

Example data grid using conditional calculations

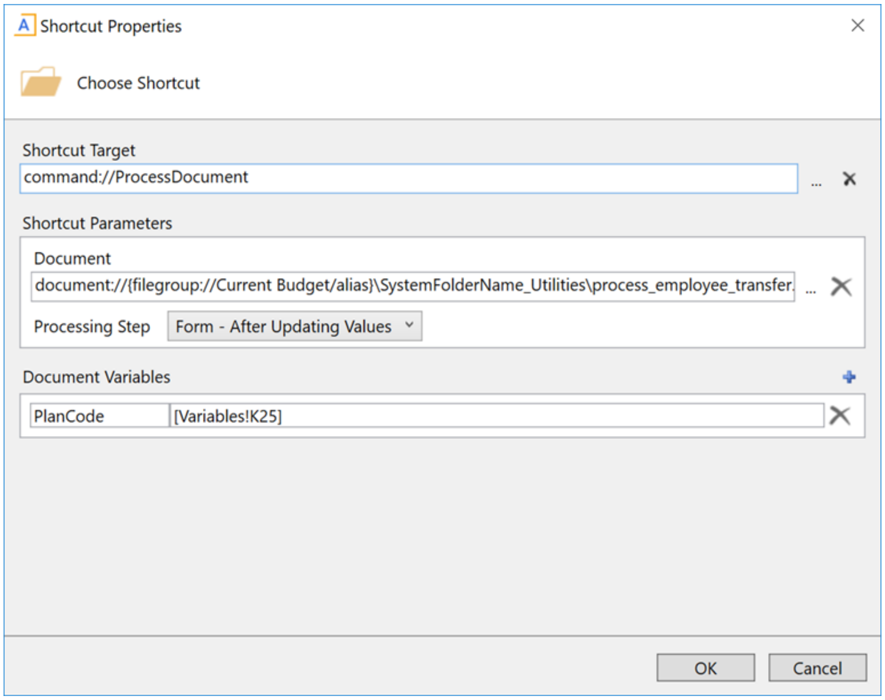
Process another document from an Axiom form

You can now initiate document processing from within an Axiom form, and have the form wait for the processing to complete before updating itself. This feature can provide improved performance for process-intensive tasks that you don't want to perform in the form itself. It can also streamline form development by allowing multiple different forms to access centralized utilities, instead of needing to duplicate functionality in the individual forms.

For example, imagine you have a form-enabled plan file to perform some payroll planning activities. When an employee is transferred from one department to another, you may have a utility that needs to be run in order to transfer the necessary data from the current plan code to the target plan code. Instead of needing to incorporate the transfer utility logic into each plan file, and instead of triggering a Scheduler task to run the utility separately, you can process the utility in the background during a form update. Because the form waits for the processing to complete before finishing the form update cycle, the updated data generated by the utility can be immediately available to the current plan file—so the user can immediately see the result of the transfer.

In order to initiate document processing from an Axiom form, use the new **Process Document** command. When you configure this command, you specify the following:

- The document to process
- The processing step during the form update, to perform the document processing during this step
- One or more document variables to pass to the document from the form (such as the current plan code, or other necessary information)



Example Process Document command

When the command is executed, the following occurs:

- The specified document is opened in the background. This process is transparent to the user.
- Active Axiom queries configured to **Refresh on Open** are processed, followed by Axiom queries configured to **Refresh During Document Processing**. (If DataLookups are present in the document, they are processed as normal depending on their configuration.)
- A save-to-database is executed in the file.

The processing occurs synchronously, during the specified processing step of the form update. Effectively, this means that the form update process waits for the document processing to complete before moving on to the next step. The data saved from the target document is available to be displayed in form components that are updated during a later processing step. For example:

- If you have a Formatted Grid that is populated by an Axiom query, you can query and display the saved data if the command is configured to execute at the **After Updating Values** processing step or earlier.

- If you have a Data Grid or a Fixed Report, you can query and display the saved data if the command is configured to execute at the **After Save Data** processing step or earlier. In this case, remember that the component used to trigger the command must be listed as a Component Dependency for the grid or report.

File groups

This section details the new features and enhancements made to file groups.

Define calc method variables using a data source

You can now define calc method variables using a data source in the file, rather than defining the variables in the calc method properties. Calc method variables are used to prompt users to define certain values, such as the account code, when inserting new rows into plan files (or other file group files).

Using a data source to define calc method variables provides the following advantages:

- Ability to dynamically change variables based on information in the file. For example, you can use formulas in the file to enable or disable variables as needed, or to define certain variable properties such as a filter for a list of accounts.
- More supported variable types such as Combo Box, Calendar, and Radio Button. The new data source supports all the same variable types as the RefreshVariables data source.
- More supported variable options such as dependent variables, multi-select, and placeholder text. However, note that some of these options are currently only supported in the Desktop Client.

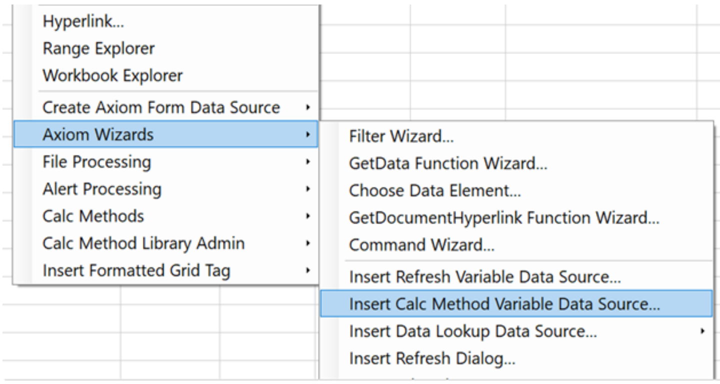
This new feature is now the preferred approach to defining calc method variables, and is intended as the focus for any new enhancements moving forward. The legacy approach is still supported, and currently there is no planned timeline to deprecate the legacy variables. However, we strongly encourage adopting the new approach for any new calc methods, or as part of any significant design changes to existing calc methods.

This section provides an overview on how to use this new feature. For more information, including design considerations and limitations, see the following topic in Axiom Software Help: *Defining calc method variables using a CalcMethodVariables data source* (AX1092).

▶ Creating the CalcMethodVariables data source

If you want to use a data source to define calc method variables, you must define a CalcMethodVariables data source in the file where the calc methods will be inserted. If the calc methods will be inserted in plan files, the data source should be defined in the template used to create the plan files. If the calc methods will be inserted into a driver file or a file group utility, the data source should be defined in that file. The data source can be defined on any sheet in the file.

To insert the data source, use the right-click menu and select **Axiom Wizards > Insert Calc Method Variable Data Source**.



New data source wizard available on right-click menu

Once the CalcMethodVariables data source has been inserted, the first step is to edit the primary tag to specify the sheet name of the calc method library and the name of the calc method. Each calc method can have an associated CalcMethodVariables data source to define the variables for that calc method, instead of defining the variables in the calc method properties.

	A	B	C	D	E	F
15						
16			[CalcMethodVariables;SHEET_NAME_HERE;CALCMETHOD_NAME_HERE]	[Name]	[DisplayName]	[VariableType]
17			[Variable]	Example Variable		String,Integer,D

Example CalcMethodVariables data source tag

For example, if the calc method is named Edit Months and the calc method library is for the Budget sheet, then the primary tag should be renamed as follows:

```
[CalcMethodVariables;Budget;Edit Months]
```

The CalcMethodVariables data source is very similar to the existing RefreshVariables data source. It supports all of the same variable types, and supports most of the same properties. All of the existing calc method variable types can be defined in the data source, as well as newly available variable types such as ComboBox, RadioButton, and Calendar.

To define a variable, create a `[Variable]` row and define the properties as needed. In this example, the data source has two variables—one to prompt the user to select an account, and another to place the associated account description within the calc method.

	A	B	C	D	E	F	G	H	I	J
3										
4		[CalcMethodVariables;Budget;Edit Months]	[Name]	[DisplayName]	[VariableType]	[RelativeLocation]	[IsRequired]	[IsEnabled]	[DependsOn]	[ColumnName]
5		[Variable]	Acct	Account	Grid	R1	TRUE	TRUE		Acct.Acct
6		[Variable]	Desc		RelatedColumnValue	S1		TRUE	Acct	Acct.Description

Example CalcMethodVariables data source with two defined variables

The main differences between the CalcMethodVariables data source and the RefreshVariables data source are as follows:

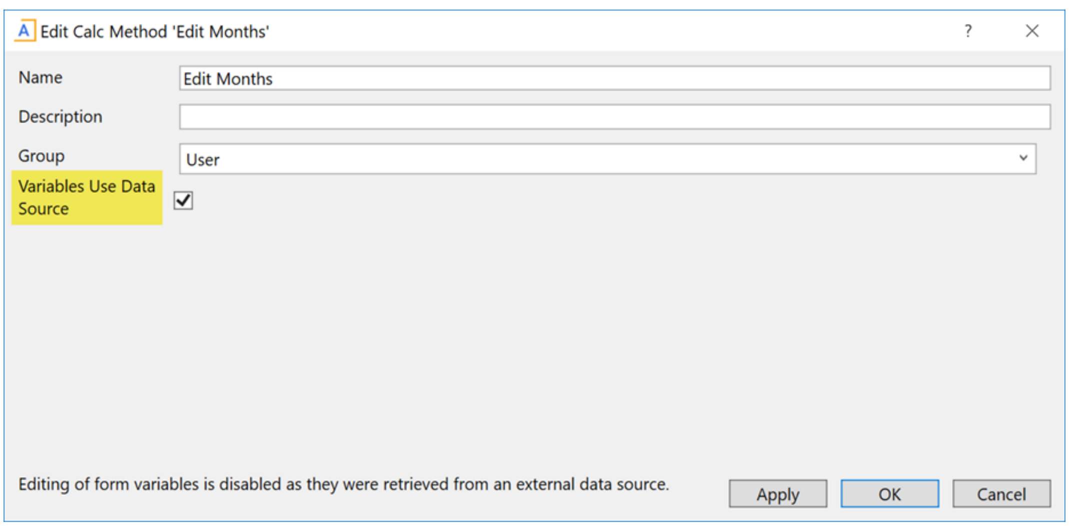
- Calc method variables use a `[RelativeLocation]` property to specify where the variable value should be placed within the calc method. This works the same way as the existing relative location property for legacy calc method variables. For example, if you specify R1, then the variable value is placed in the first row of the calc method, in column R.

- Calc methods variables also use an `[InsertOnly]` property to specify whether the variable only applies when the calc method is inserted. This works the same way as the existing insert only property for legacy calc method variables.
- The `[SelectedValue]` property is still present in the CalcMethodVariables data source, but it only applies when configuring dependent variables for use in the Desktop Client. During the calc method insertion process, the user's selected value for the parent variable is temporarily written back to the selected value field, for purposes of resolving any dynamic formulas used in the child variable.
- Calc method variables do not support the ability to define a default value, so all properties relating to setting or clearing a default value are not present in the CalcMethodVariables data source. Other inapplicable properties, such as the ability to define a group name, are also not present.

Configuring the calc method to use the data source

For each calc method, you can choose whether to use a data source to define variables, or use the legacy approach. This is specified in the calc method properties, and can be done when creating the calc method or when editing the properties of an existing calc method.

To use a data source, select the new option **Variables Use Data Source** in the calc method properties:



Example Edit Calc Method dialog with new option to use a data source

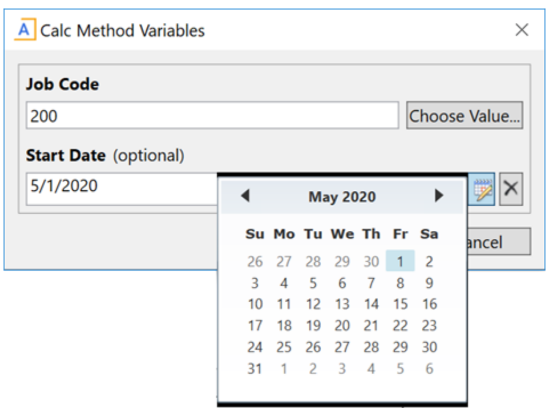
When this option is selected, the section at the bottom of the dialog—where legacy calc method variables are defined—becomes hidden. By default, this option is not selected, so that all existing calc methods will continue to use any defined legacy variables.

User experience

When a calc method is configured to use a CalcMethodVariables data source, the basic user experience is the same as when using legacy calc method variables—meaning, a Calc Method Variables dialog displays to prompt the user for their variable selections. The difference is that the variables are rendered based on the data source defined in the current file instead of based on the calc method properties stored in the calc method library.

Because the variables are defined in the file using the CalcMethodVariables data source, the variables presented to the user can dynamically change based on information in the file. For example, certain variables could be enabled or disabled based on some condition defined in the file. The filter used to present a list of accounts or a list of job codes could also dynamically change.

Additionally, because the new CalcMethodVariables data source is modeled on the existing RefreshVariables data source, several new variable types are now supported and will display in the Calc Method Variables dialog (in either the Desktop Client or the Web Client). For example, when inserting a calc method for a new employee, you might want to prompt the user to enter a start date from a Calendar variable.



Example Calendar calc method variable

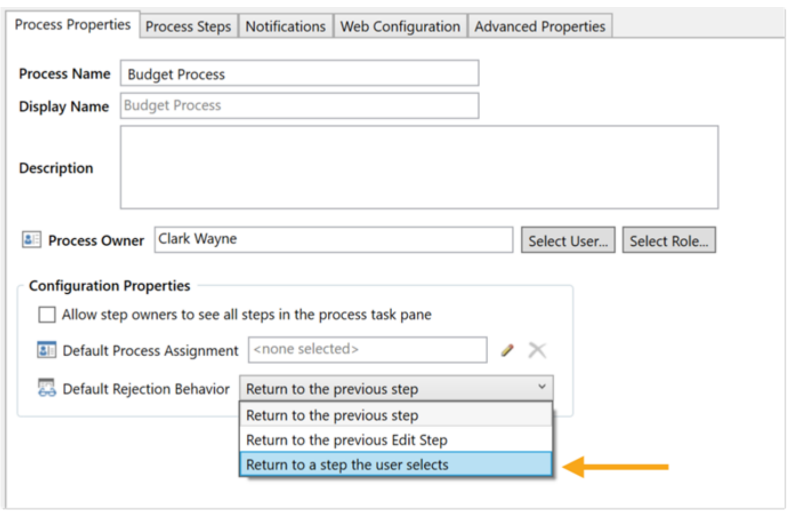
Plan file processes

This section details the new features and enhancements made to plan file processes.

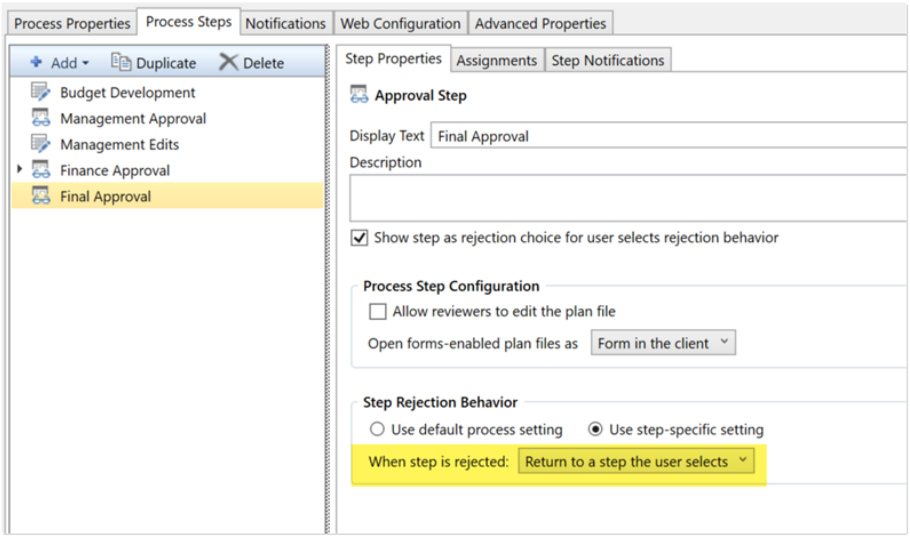
Reject plan files back to a user-selected step

You can now allow users to select which step they want a plan file to return to, when rejecting an approval step in a plan file process. This provides approving users with the flexibility to return the plan file back to the proper place in the process.

To provide this functionality, an additional option for rejection behavior is now available in the plan file process definition. If you want to allow users to select the return step for the rejection, then set the rejection behavior to **Return to a step the user selects**. This option can be enabled on a per step basis for Approval Steps and Multiple Approvals Steps, or it can be set at the process level to provide the default rejection behavior.

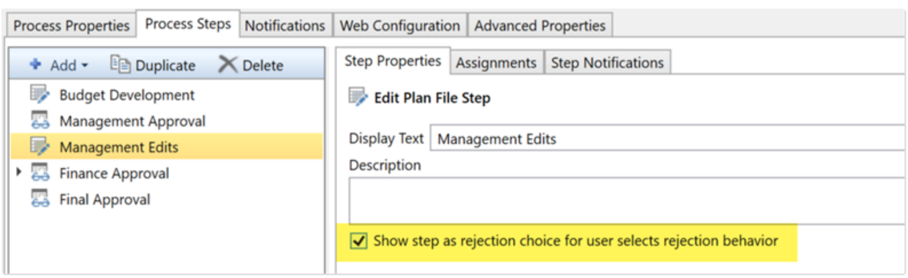


New option for process-level default rejection behavior



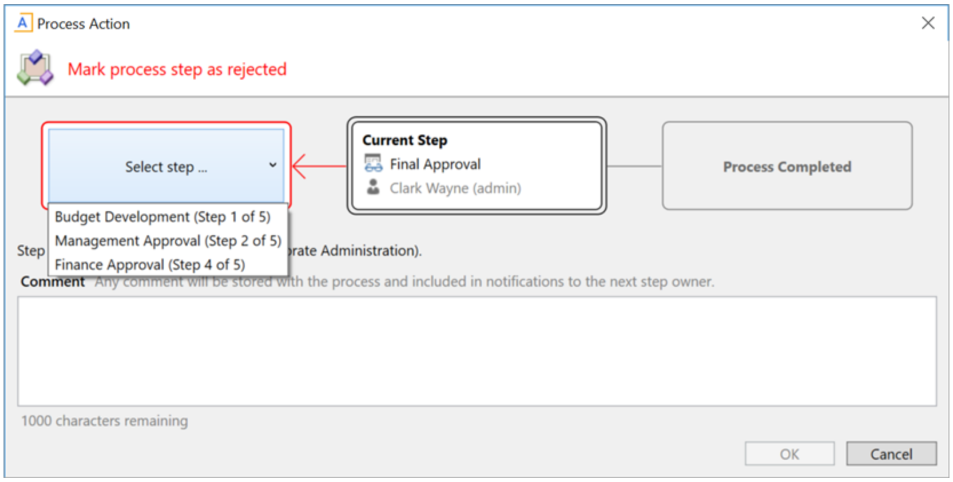
New option for step-level rejection behavior

When using this rejection option, you may only want a subset of steps to be eligible as the return step for the rejection. For any step that you do not want to be available, you can clear the option **Show step as rejection choice for user selects rejection behavior**. By default this option is enabled for all steps, so all steps are eligible as return steps.

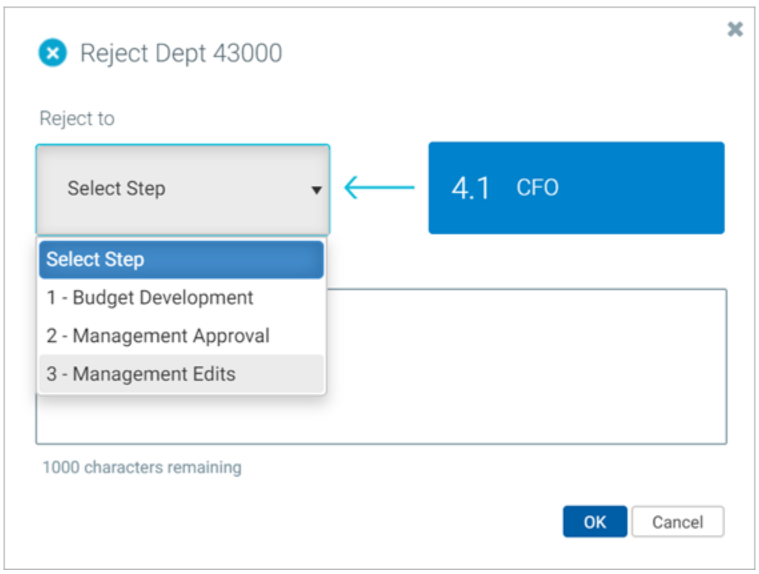


New option to include or exclude steps for user-selected rejection behavior

If you are the owner of a step that is configured for user-selected rejection, and you choose to reject the step for the plan file, then you are prompted to select the return step for the rejection. You can select any step that comes before the current step, as long as **Show step as rejection choice for user selects rejection behavior** is enabled for that step. For example, you might choose to reject a plan file all the way back to the beginning of the process, if the reason for the rejection requires the step to move through these steps again.



Example user-selected rejection step in the Desktop Client

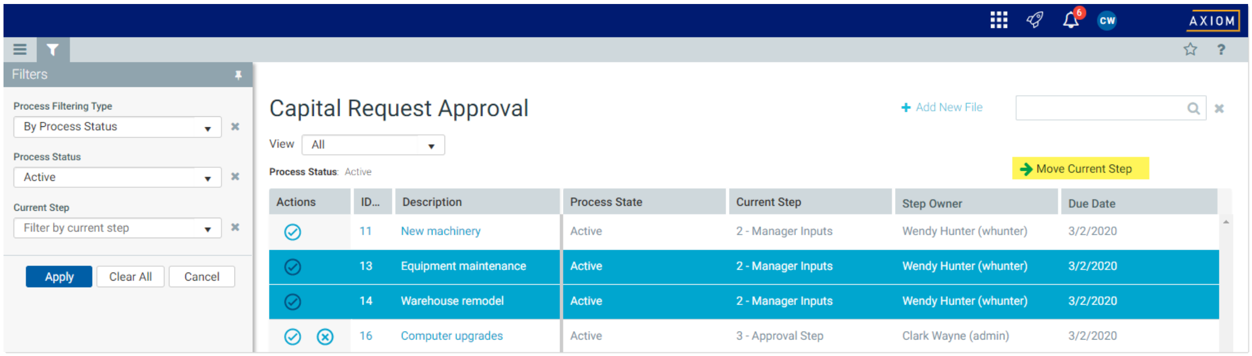


Example user-selected rejection step in the Web Client

Move plan files to different steps using the Web Client

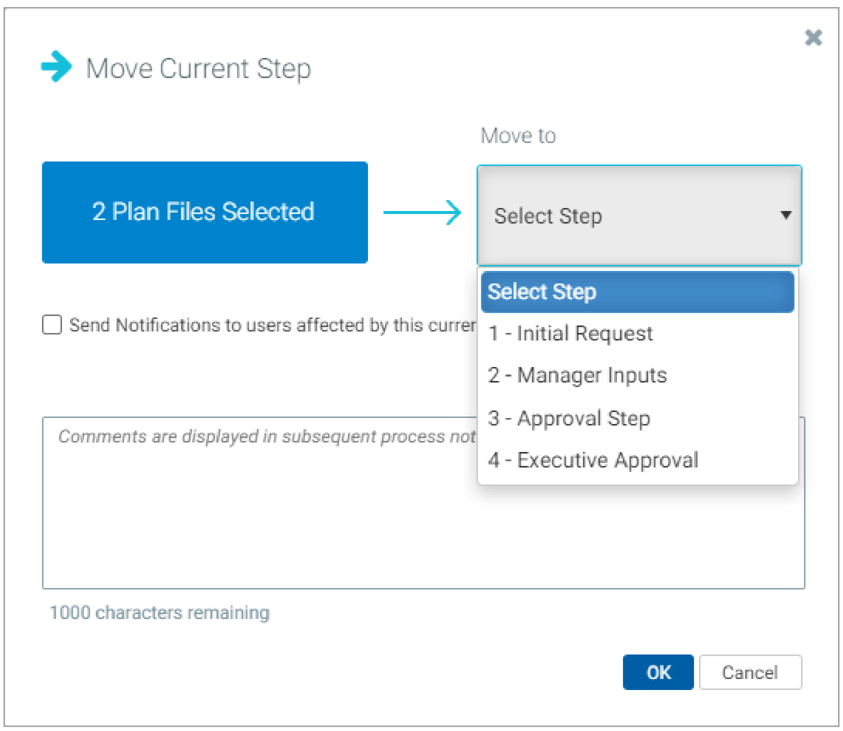
Administrators and process owners can now use the Web Client to move plan files to different steps in a plan file process as needed. Previously this functionality was only available in the Desktop Client, using the Process Status dialog. It is occasionally necessary to move plan files in order to make administrative adjustments to a process.

On the Process Directory page, you can move one or multiple plan files to a different step. Select the row or rows for the plan files that you want to move, then click **Move Current Step** in the top right.



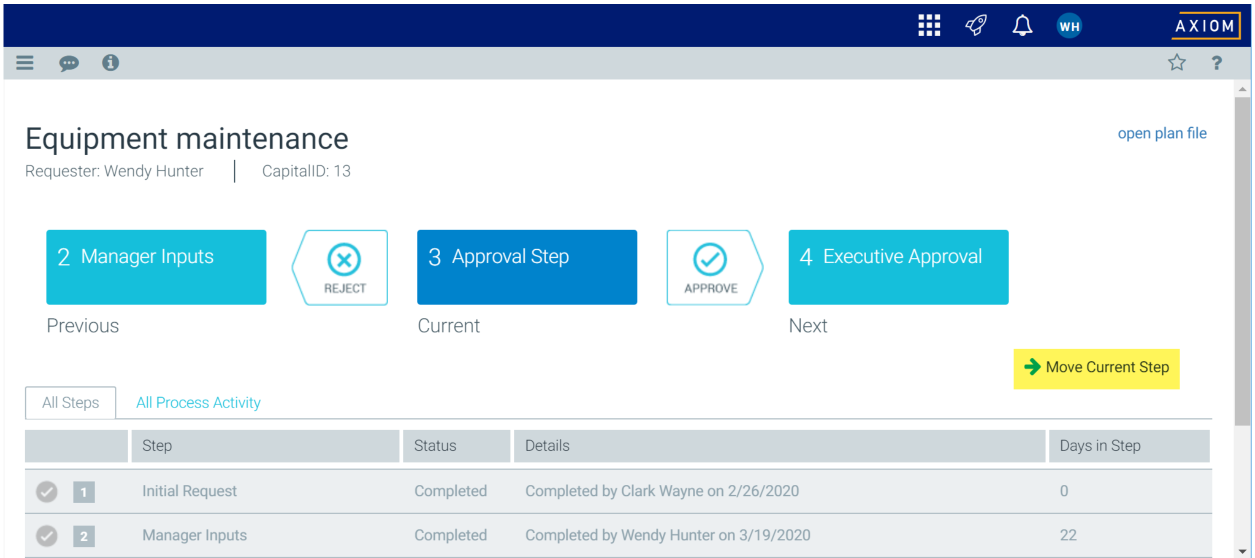
Example Move Current Step option in Process Directory

In the **Move Current Step** dialog, use the **Select Step** drop-down to select the step that you want the plan files to move to. You can also optionally send a notification to the new step owners. When you click **OK**, the plan files are aborted in the current step and moved to the new step.



Example Move Current Step dialog

Similar functionality is now also available on the Process Routing page, to move the current plan file to a different step.



Example Move Current Step option on Process Routing page

The Move Current Step option does not display to end users; it is only visible to administrators and process owners.

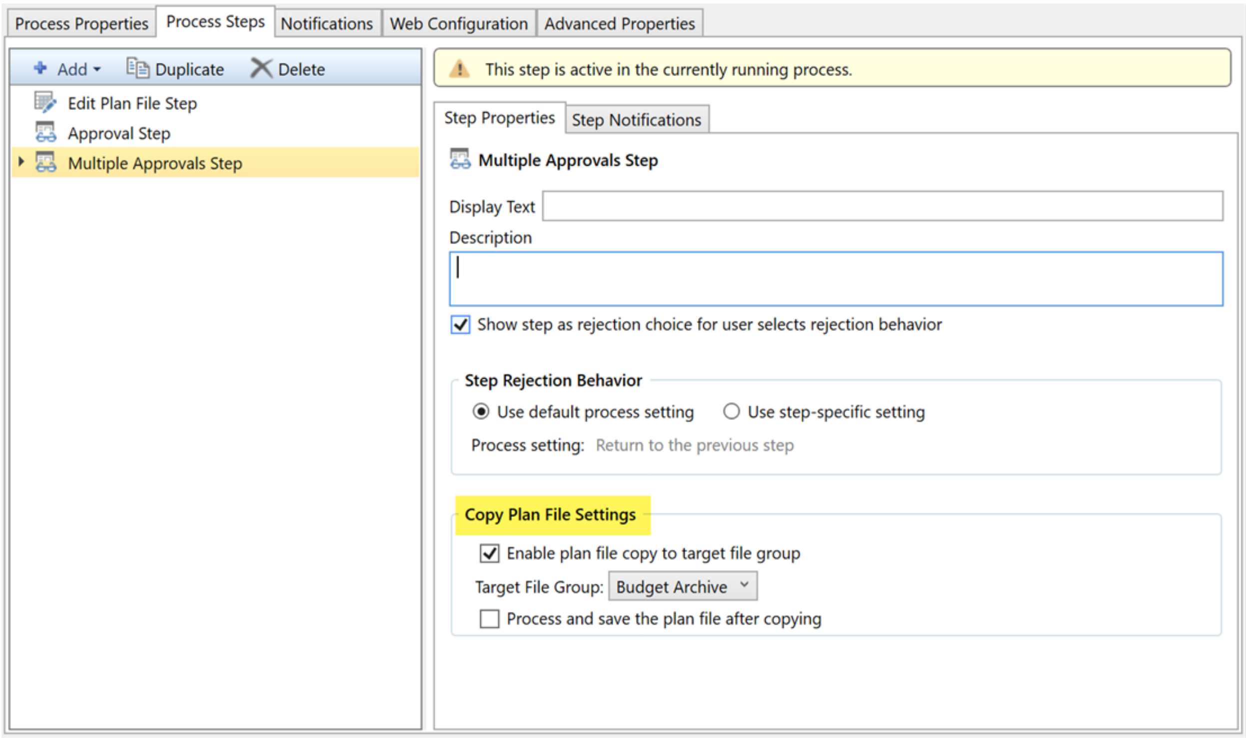
Copy plan files after completing a process task

You can now configure a step in a plan file process to copy plan files from one file group to another when the step is completed. This feature can be used to support file group versioning in conjunction with a plan file process.

▶ Configuring a process step to copy plan files

Because this feature only applies to very specific use cases, it is hidden by default. If you want to use this feature, you must first set the system configuration setting **ShowCopyPlanfilesActionInProcess** to **True**. For information on editing system configuration settings, see the following topic in Axiom Software Help: *System Configuration Settings* (AX2015).

When this feature is enabled, a section titled **Copy Plan File Settings** becomes available on Approval Steps and Multiple Approvals Steps.



New process option to copy plan files

NOTE: This feature can only be used to copy plan files between standard file groups. If the plan file process is for an on-demand file group, the option does not display, even if the system configuration setting is enabled.

If you want a step to trigger a plan file copy action, complete this section as follows:

- Select **Enable plan file copy to target file group**.
- Select the target file group for the copy action.
- If you want the copied plan file to be saved and processed within the target file group, select **Process and save the plan file after copying**. If you do not select this option, then the plan file will be copied to the target file group but its data will not be reflected in the associated tables for the target file group. You might want to copy without processing if the target file group is only intended as an archive, or if plan files in the target file group get processed in some other way.

When this step is approved and the plan file moves to the next step in the process, the plan file is automatically copied to the target file group. If the processing option is enabled, then the newly copied plan file is also processed by Scheduler. This includes refreshing all active Axiom queries where **Refresh during document processing** is enabled, executing a save-to-database, and then saving the file.

For Multiple Approvals Steps, the copy action can only be enabled on the parent step, not on the child approval steps. If enabled, the copy action occurs when all child approval steps have been completed, and any processing is performed as the user who completed the last child approval step.

The copy action is not performed if the step is rejected instead of approved, or if the step is aborted or skipped. The plan file must complete the step in order to trigger the copy action.

For more information on how this feature works, including requirements and limitations, see the following topic in Axiom Software Help: *Copying plan files when a process step is completed* (AX1123).

Additional enhancements

- When viewing a plan file's process status on the Process Routing page, you can hover your cursor over the **Details** column in the **All Steps** tab to see the most recent comment associated with each step.
- New file diagnostics tests and utilities have been added to the **Run QA Diagnostics** command. These tests can be used to review your files and check for errors or potential design issues.
  - **Check Document Links**: This test scans the document for invalid links to other documents—for example, if the Associated Task Pane field in the Control Sheet points to an invalid file path. When using the **Run Bulk QA Diagnostics** utility, this test can also be run on the following additional file types: Scheduler jobs, imports, exports, task panes, and ribbon tabs.
  - **Search Documents**: This utility can be used to search the documents in a specified directory for a specified string. This can be useful if you need to rename an asset and you want to find references to the old name, or if you want to look for usage of a particular asset.