Axiom Function Reference
Axiom Software
Version 2018.1

KaufmanHall
axiom software

Table of Contents

Function Overview

Axiom functions can be used to bring Axiom Software data or information into specific cells of a spreadsheet file. Some Axiom functions can be used to perform an action.

Axiom functions can be used in any spreadsheet opened within Axiom Software. Axiom functions are Microsoft Excel user-defined functions (UDFs).

> **NOTE:** Whenever possible, Axiom queries or data lookups should be used instead of Axiom functions, for performance reasons. An Axiom function should only be used when it is the only way to return the desired information. For more information, see the *Axiom File Setup Guide*.

▶ Intended audience

This guide is intended for administrators and other power users who are responsible for building files and using Axiom functions.

▶ What is covered in this guide?

This guide serves as a reference for all available functions in Axiom Software:

Function	Description
GetCalcMethod	Inserts a calc method at the current location, using the row's InsertCM tag if defined.
GetColumnInfo	Returns information about a table column, given the column name.
GetColumnLetter	Returns a column letter based on a specified context.
GetCurrentSheetView	Returns the names of the currently applied views for a specified sheet.
GetCurrentValue	Returns information about the current filter context applied to the file, for either multipass processing or for a temporary sheet filter.

Function	Description
GetData	Returns data from a table. Can be used to return summed data from data tables, or specific data records from reference tables and document reference tables.
GetDataElement	Launches the Choose Value dialog and places the user's selected value in a designated cell.
GetDocument	Opens a specified file when the cell containing the function is double-clicked.
GetDocumentHyperlink	Returns a hyperlink to a file in the Axiom Software file system, that can be used to launch the system and open the file.
GetDocumentInfo	Returns information about the current document, such as the file name, the folder name, or the document ID.
GetFeatureInfo	Returns information about features installed by a package.
GetFileGroupID	Returns the ID of the current file group or a specified file group.
GetFileGroupProperty	Returns information about the current file group or a specified file group.
GetFileGroupVariable	Returns the value of a variable defined for the current file group or a specified 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.
GetFormDocumentLinkTag	Returns a content tag that will render as a hyperlink to an Axiom form in the Axiom file system, when used within a form.
GetFormDocumentURL	Returns a URL to a form-enabled document in the Axiom file system, to open an Axiom form from within another form.
GetFormResourceLinkTag	Returns a content tag that will render as a hyperlink to a non-Axiom file in the Axiom file system, when used within an Axiom form.
GetFormResourceURL	Returns a URL to a non-Axiom file in the Axiom file system, to open the file in its native application from within an Axiom form.
GetFormState	Returns the current form state value for a specified form state key. Also can be used to set the default value if no value currently exists.
GetPeriod	Returns the current period for the system or for a table.

Function	Description
GetPlanFileAttachment	Opens a specified plan file attachment or a dialog to allow the user to select an attachment to open.
GetPlanFileDocumentID	Returns the document ID for a particular plan file within a file group.
GetPlanFilePath	Returns the file path for a particular plan file within a file group.
GetProcessInfo	Returns information about a process, such as the due date for a step or a step owner.
GetRowNumber	Returns a row number based on a specified context.
GetSecurityInfo	Returns information about a user's security settings, such as the file group filter, the table type filter, or individual security permissions.
GetSharedVariable	Returns the current value for a specified shared variable. Also can be used to set the initial value for the variable if no value currently exists.
GetSystemInfo	Returns information about the current Axiom Software system, such as the system name or the product version.
GetTableInfo	Returns information about a table, given the table name.
GetURLEncodedString	Returns an encoded string that is valid for use in a URL, given a string with spaces or other special characters.
GetUserInfo	Returns information about a user, such as the user ID, the user name, and the user email.
GetWebReportDocumentURL	Returns a URL to a web report in the Axiom file system, to open the web report from another web report or an Axiom form.
IsRunningMultipass	Returns True if multipass processing is currently occurring; returns False if not.
ManagePlanFileAttachments	Opens a dialog to allow the user to add, edit, or delete a plan file attachment for a specified plan file.
RunEvent	Runs Scheduler jobs that contain a specified event.
SetSharedVariable	Sets and returns the value for a specified shared variable, so that the value can be referenced by other forms within the shared form context (using GetSharedVariable).
ShowFilterWizardDialog	Opens the Filter Wizard dialog to allow the user to create a filter. The filter is then placed in a designated cell.
ShowFormDialog	Opens a designated form-enabled file as a dialog.

Action Functions

This section contains information on Axiom functions that allow you to perform an action within an Axiom spreadsheet file. When the cell containing the function is double-clicked, the associated action for the function occurs—such as opening a specified document or inserting a calc method at the current location.

GetCalcMethod function

Inserts a calc method at the current location. If standard calc method controls are used in the sheet (InsertCMColumn), then the current row's InsertCM tag is used to control the insertion. If calc method controls are not used in the sheet, the standard Insert Calc Method dialog displays.

To insert the calc method, the user double-clicks on the cell containing the GetCalcMethod function.

> **NOTE:** GetCalcMethod is not for use in Axiom forms. If you want to allow users to insert calc methods within Axiom forms, you must use the Add Rows command (either in a Button component or as a tag within a formatted grid). For more information, see the *Axiom Forms and Dashboards Guide*.

▶ Syntax

```
GetCalcMethod("DisplayText")
```

Parameter	Description
DisplayText	The display text for the cell containing the RunEvent function. For example, the text could be something like "Insert a new account."
	The DisplayText displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- GetCalcMethod is not supported for use with dynamic calc method controls (DynamicCMColumn). If a DynamicCMColumn is present in the sheet, then GetCalcMethod does not allow calc method insertion, even if the current row happens to have an InsertCM tag. However, if the sheet uses both dynamic controls and standard controls, then GetCalcMethod will work as expected with the enabled rows in the InsertCMColumn.
- All calc method security permissions are honored as normal. The user must have rights to insert a calc method in order to use GetCalcMethod. If the user does not have the appropriate rights, then an error message displays when the function is double-clicked.
- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use GetCalcMethod.
- GetCalcMethod is only valid for use in files that support calc method libraries. If used in other files, an error message displays when the function is double-clicked.
- GetCalcMethod is a non-volatile function.

▶ Examples

=GetCalcMethod("Insert a new account")

When a user double-clicks on the cell containing this function, Axiom Software checks for the presence of an InsertCMColumn in the sheet.

- If an InsertCMColumn exists, Axiom Software checks for the presence of an InsertCM tag on the row:
  - If an InsertCM tag exists, that tag is used to control the calc method insertion. For example, if insertion is enabled and only one calc method is allowed, that calc method is inserted. If the tag does not allow insertion, then an error message is displayed.
  - If an InsertCM tag does not exist, then insertion is not allowed and an error message is displayed.
- If no InsertCMColumn exists, the default Insert Calc Method dialog displays. The selected calc method is inserted.

GetDataElement function

Launches the Choose Value dialog where the user can select a value (such as an account number), and places the selected value in a designated cell. The function parameters determine which values are available for selection in the dialog.

To launch the wizard, the user double-clicks on the cell containing the GetDataElement function.

> **NOTE:** GetDataElement is not for use in Axiom forms. If you want users to be able to select values from within an Axiom form, you can use the Combo Box component or a Formatted Grid component (Select tag). For more information, see the *Axiom Forms and Dashboards Guide*.

▶ Syntax

```
GetDataElement("DisplayText", "TableColumn", "TargetCell", "Filter",
"AdditionalColumns", "UseDataSource", "HideKey")
```

Parameter	Description
DisplayText	The display text for the cell containing the GetDataElement function.
	For example, the text could be something like "Select an account" or "Double-click to select an account."
	The DisplayText displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.
TableColumn	The table column from which the user will select a value. You must use fully qualified Table.Column syntax, or use an alias name.
	For columns with lookups, the specified table impacts the list of items to be returned. For example, `GL2018.Dept` returns only the departments used in the GL2018 table, whereas `Dept.Dept` returns the full list of departments defined in the Dept table.
	NOTE: If the specified column uses multi-level syntax, then the resulting list will be sourced from the final lookup table, not the original table. For example, if you specify `Encounter.PrimaryPhysician.FirstName`, then the list will contain all first names from `Physician.FirstName`, instead of only the first names for the physicians used in the Encounter table.
	If the value column is a key column in a reference table (such as ACCT.ACCT), then the descriptions are automatically displayed in the dialog if the table has a designated description column.
	Alternatively, you can specify the name of a DataElements data source defined within the file. To do this, you must also set the value of the UseDataSource parameter to `TRUE`.

Parameter	Description
TargetCell	Optional. The target cell in which to place the selected value.
	If no target cell is specified, then the selected value is placed in the cell containing the GetDataElement function. This value replaces the function, meaning that it is no longer possible to launch the Choose Value dialog from this location.
	NOTE: Remember to place the target cell location in quotation marks, such as "C25". If you omit the quotation marks and use just C25, then the entry is interpreted as a cell reference within the function—meaning that Axiom Software will attempt to read the target cell location from the cell reference. If C25 is empty, the function will behave as if no target cell was specified. If C25 contains content other than a cell location, the function will return an error when it is double-clicked.
Filter	Optional. A filter criteria statement to limit the list of values displayed in the Choose Value dialog. Use standard filter criteria syntax.
	This parameter does not apply if you are using a DataElements data source.
AdditionalColumns	Optional. One or more columns in the source table, or a related table, to display along with the value column for information purposes. Only applies if the value column for the variable is the key column of a reference table. Separate multiple column names with commas.
	For additional columns in the source table, you can list column-only names, or use Table.Column syntax—for example, you can list `Region` or `Dept.Region`. If you want to use a column from a related table, you must use Table.Column syntax. When you use Table.Column syntax, the column header in the dialog displays exactly as typed in the function, so be sure to use title case if you want the header to display that way.
	NOTES:
	• The designated description column is always included if the value column is a key column of a reference table; you do not need to list it in this parameter.
	• If the value column is from a data table with multiple key columns, then additional columns cannot be included.
	• This parameter does not apply if you are using a DataElements data source.

Parameter	Description
UseDataSource	Specifies whether the function should treat the second parameter of the function as a table column name or as a DataElements data source. • If FALSE (default), then the second parameter is treated as a table column name. If the entry does not match a table column name defined within Axiom Software, the function returns an error. • If TRUE, then the second parameter is treated as a DataElements data source name. The fourth and fifth parameters are ignored. For more information on using a DataElements data source, see Using a DataElements data source with GetDataElement.
HideKey	Specifies whether the key column is hidden in the dialog. Only applies when the specified value column is a key column of the table. • If FALSE (default), then the key column is shown in the dialog. • If TRUE, then the key column is hidden in the dialog. Users make their selection based on other columns in the table, and then the key column value is inserted into the designated cell. This is intended for situations where the user needs to select a value from the key column, but the code that users recognize is held in a different column of the table.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- In cases where a target cell is specified, the current value in that cell is automatically selected in the list when the Choose Value dialog is launched.

- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use GetDataElement.

- The Choose Value dialog uses either the full Axiom grid format or a "simple view" format depending on the information shown in the dialog. If only one column is shown (including concatenated descriptions, if the column is a key column), then the simple view is used. If multiple columns are shown, then the full grid view is used.

  The number of records available to the dialog depends on the view being used. Simple view is limited to showing 5,000 records, but the search box can be used to find any record returned by the query. Full grid view is limited to showing 10,000 records by default, for both display and search (this limit can be adjusted using the system configuration setting **MaxChooseValueRows**).

- GetDataElement is a non-volatile function.

▶ Examples

All of these examples use a table column. For an example of using a DataElements data source, see the following section.

```
=GetDataElement("Select an account","ACCT.ACCT")
```

This example opens the Choose Value dialog in "simple view" and displays all values of the ACCT.ACCT column. Once a value is selected, it is placed in the current cell, replacing the GetDataElement function.

```
=GetDataElement("Select an account","ACCT.ACCT","C25",
"Acct.Category='Revenue'")
```

In this example, the optional filter parameter is used to filter the list of ACCT values to only show Revenue accounts. Additionally, the selected value will be placed in cell C25 instead of the current cell. This GetDataElement function can continue to be used to change the account value placed in cell C25.

```
=GetDataElement("Select a
department","DEPT.DEPT",,,"VP,Region,Region.RegionType")
```

This example opens the Choose Value dialog in full grid view, in order to display the additional columns specified in the fifth parameter. The columns VP and Region are on the target DEPT table, whereas Region.RegionType is used to bring in a column from a related table.

```
=GetDataElement("Select a region","DEPT.Region","C25")
```

This example uses a grouping column instead of a key column, so no additional columns can be specified. The dialog will always open in simple view, displaying the list of regions.

```
=GetDataElement("Select a
request","Request.RequestID","C25",,"Request.RequestCode",False,True)
```

In this example, the source column is being hidden using the HideKey parameter. The key column is an identity column, so the RequestCode column contains a more meaningful code that is built up for each item in the table. Users will use the RequestCode to select an item from the dialog, and then the appropriate key column value will be written to the specified cell.

▶ Using a DataElements data source with GetDataElement

To define a custom list of values for use with GetDataElement, you must create a DataElements data source within the file. This data source works the same way as data sources for Axiom form components. The data source has a primary tag to identify the control row and control column, and then a series of row and column tags to flag the data.

The tags for the DataElements data source are as follows:

Primary tag

[DataElements;*DataSourceName*;*DialogTitle*]

> *DataSourceName* defines the name to be placed in the second parameter of the GetDataElement function. For example, if the data source defines a list of available status

values, the name might be Status.

DialogTitle is optional, and defines text to display in the title bar of the GetDataElement selection dialog. If this parameter is omitted, the title of the dialog is "Choose Value."

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

[Header]

The Header row contains the text to display as the header of each column in the GetDataElement selection dialog.

[Row]

Each row flagged with this tag defines an item that can be selected in the GetDataElement selection dialog.

Column tags

[Column]

Each column flagged in the data source will display in the GetDataElement selection dialog, using the header title displayed in the Header row. For example, you might have one column for status names and one column for descriptions of the status values.

[Value]

The Value column contains the value that will be placed in the target cell for GetDataElement.

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 is an example DataElements data source:

◢	A	B	C	D	E
1					
2		[DataElements;Status;Select Status]	[column]	[column]	[value]
3		[header]	Status	Description	
4		[row]	Approved	The request has been approved	Approved
5		[row]	Denied	The request has been denied	Denied
6		[row]	On Hold	The request is on hold, awaiting further information	On Hold
7		[row]	Cancelled	The request has been cancelled	Cancelled

You can type the data source tags manually, or use the data source wizard. To use the wizard, select any list items that you have already entered into the spreadsheet, then right-click and select **Axiom Wizards > Create Data Element Data Source**. You can also right-click an empty cell to add the tags first and then enter the list items.

To use a DataElements data source, the GetDataElement function should be set up as follows:

- The second parameter must indicate the data source name instead of a table column name.
- The fourth and fifth parameters do not apply and should be left blank.
- The sixth parameter must be set to True. This tells Axiom Software to use a data source instead of a table column.

For example, the GetDataElement function might look as follows:

```
=GetDataElement("Select a status","Status","F24",,,True)
```

When a user double-clicks the GetDataElement cell, the selection dialog displays as follows:
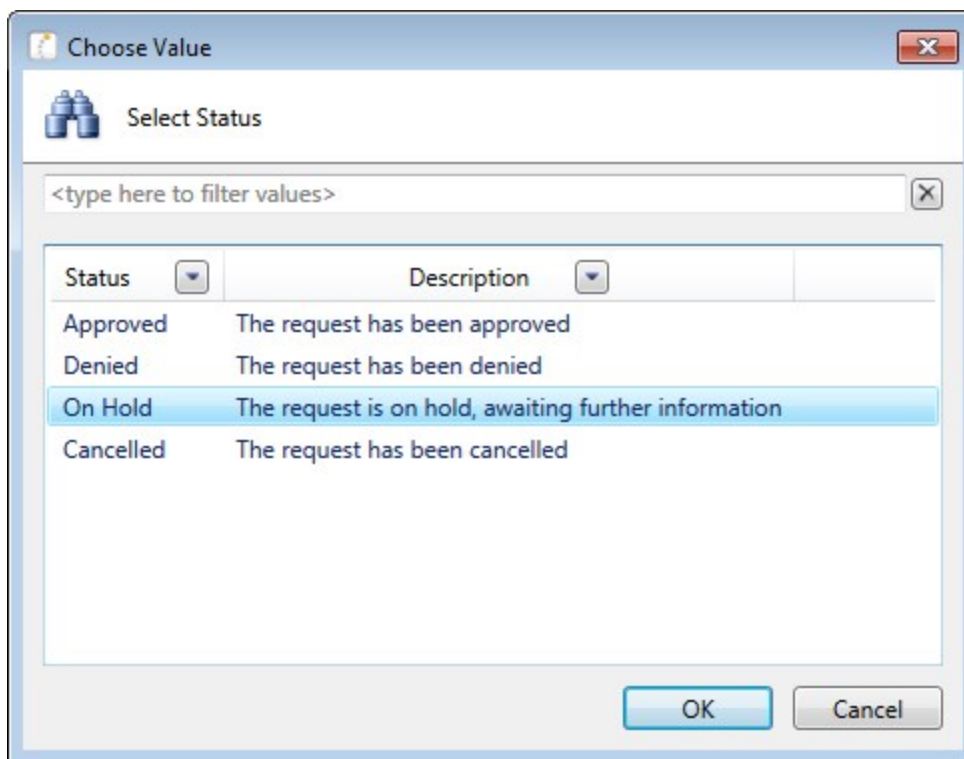
When a user selects an item in the dialog and clicks OK, the corresponding value from the Value column in the data source is placed in the target cell.

> **NOTE:** When using a data source, the Choose Value dialog is always in full grid view instead of simple view.

GetDocument function

Opens a file in the Axiom Software file system—for example, a plan file or a report. Users can double-click on the cell that contains the function to open the file.

> **NOTES:**
> - The GetDocument function is intended to be used within Axiom files, to open other Axiom files or forms. If instead you want to create an externally usable hyperlink that can both launch the system and open the file, use the GetDocumentHyperlink function.
>
> - The GetDocument function cannot be used within Axiom forms. If you want to let users open other Axiom files or forms from within an Axiom form, you must use other means. For more information, see the Axiom Forms and Dashboards Guide.

▶ Syntax

```
GetDocument("DisplayText", "DocumentPathorID", "SheetName", "CellAddress",
CloseOnNavigate, OpenReadOnly, ForceReopen, ViewAsForm, "WebTabName",
"Variables")
```

Parameter	Description
DisplayText	The text to display in the cell.
	The DisplayText displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.

Parameter	Description
DocumentPathorID	Optional. The file to open when the user clicks on the cell. Specify either the document ID, or the full path to the file.
	The behavior for this parameter is as follows:
	- If omitted (blank), the current file is assumed. This could be used to jump from one sheet in a file to another sheet in the same file.
	- If a document ID is specified, or a complete folder path plus file name, that specific file is opened.
	- If a folder path is specified with no file name, then all files in that folder are opened. You must place a backward slash at the end of the folder path, such as: "\Axiom\Reports Library\North Reports\"
	- If a folder path is specified with a file filter (a file name that uses wildcard characters), then all files that match the filter are opened. For example: "\Axiom\Reports Library\Misc Reports\North*.xlsx" to return all files that start with North.
	When multiple files are opened, the last file opened will be the active document, and the sheet and cursor position will be as they were when the file was last saved. The SheetName and CellAddress parameters are ignored in this circumstance.
SheetName	Optional. The name of the sheet to make active when the file is opened. If omitted, the file opens to the sheet that was active when the file was last saved.
	This parameter only applies when the DocumentPath specifies a single file. Otherwise, it is ignored.
CellAddress	Optional. The cell location to make active when the file is opened. If omitted, the file opens to the cell that was active when the file was last saved.
	This parameter only applies when the DocumentPath specifies a single file. Otherwise, it is ignored.
CloseOnNavigate	Optional. A Boolean value that determines how the current file is treated when the target file is opened.
	- If FALSE, then the current file remains open. False is the default value if this parameter is not specified.
	- If TRUE, then the current file is closed.

Parameter	Description
OpenReadOnly	Optional. A Boolean value that determines whether the target file is opened as read-only. • If FALSE, then the target file opens with the user's normal level of access rights as defined in Security. False is the default value if this parameter is not specified. • If TRUE, then the target file is opened as read-only, regardless of whether the user has a higher level of access to the file.
ForceReopen	Optional. A Boolean value that determines whether the target file is reopened if it is already open when a user double-clicks the GetDocument cell. • If FALSE, then the target file is not reopened if it is already open. False is the default value if this parameter is not specified. • If TRUE, then the target file is closed and then reopened if it is already open. You might set this to True if the target file uses Axiom queries that are set to refresh on open, and you want the refresh to occur automatically whenever the user double-clicks on GetDocument.
ViewAsForm	Optional. A Boolean value that determines whether the target file is opened as an Axiom form. • If FALSE, then the target file is opened as the source spreadsheet for the Axiom form. False is the default value if this parameter is not specified. • If TRUE, then the target file is opened as an Axiom form. By default, the Axiom form will open in the user's browser unless a WebTabName is also specified. This parameter only applies to form-enabled files. It is ignored otherwise.
WebTabName	Optional. The name to display on the file tab within the Axiom Excel Client or Windows Client, when the target file is opened as an Axiom form. If a tab name is defined, then the Axiom form will open as a web tab within the application. If this parameter is not specified, then the Axiom form will open in the user's browser. This parameter can also optionally be used when opening non-form Axiom files, to specify an alternate tab name for the file.

Parameter	Description
Variables	Optional. One or more variable / value pairs to pass to the target file when it is opened. Variable / value pairs are specified as follows:

<div align="center">

VariableName=Value;*VariableName=Value*

</div>

Separate multiple variable / value pairs using semicolons. If a variable value contains a semicolon, then it must be preceded by a backslash (\) so that Axiom does not treat the semicolon as a delimiter. Equals signs within a value (such as to pass a filter criteria statement as a value) do not need to be specially treated.

Variable values can be read in the target file by using the GetDocumentInfo function. For more information and examples, see Passing values from one file to another.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- The user's security settings are honored to determine if the user has the right to open a specified file. If the user cannot access any documents specified by the DocumentPathorID, then a message displays to the user to inform them that they do not have permission to access the file(s). If a folder path is specified (with or without a file filter), and the user has access to some documents in the folder, then only those documents are opened, and no message displays to the user about the other documents that could not be opened.

- If you specify a folder as the DocumentPathorID (with or without a file filter), only files directly within the folder will be opened—subfolders are ignored.

- Use caution when specifying a folder as the DocumentPathorID (with or without a file filter). If opening all files in the specified folder results in a large number of files, Axiom Software will attempt to open them all, which may take a long time (and could potentially cause your client to run out of memory). A folder path should only be specified if it will result in a small, targeted number of documents.

- To obtain a file path for use in the function, you can right-click a file in Axiom Explorer or the Explorer task pane and choose **Copy document path to clipboard**, then paste the path into the function parameters (or into a cell that you will reference).

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use

GetDocument.

- GetDocument is a non-volatile function.

▶ Examples

```
=GetDocument("Open Income Statement","\Axiom\Reports Library\Misc
Reports\Income Statement.xlsx","Report","E12")
```

This example opens the specified income statement report, with the Report sheet active and with the cursor at cell E12.

```
=GetDocument("Open Income Statement",93,"Report","E12")
```

This example is the same as the prior example, except the document ID is used instead of a folder path and file name.

```
=GetDocument("Open Plan",GetPlanFilePath("Budget 2019",3000),,,True)
```

This example opens the specified plan file, to whichever sheet and cell were active when the file was last saved. The file that contains the GetDocument function is closed when this file is opened. The separate function GetPlanFilePath is used to return the path of the plan file.

```
=GetDocument("Open North Region Reports","\Axiom\SystemFolderName_
ReportsLibrary\Regional Reports\North*.xlsx",,,,True)
```

This example opens all reports in the Regional Reports folder where the file name starts with "North." This example uses the system folder name syntax, so the files in the specified folder can be opened regardless of what language the end user is running the system in. This example also uses the OpenReadOnly optional parameter to specify that the file is opened as read-only.

```
=GetDocument("Payroll Sheet",,"Payroll")
```

This example omits the document path, so that the current document is assumed. It jumps to the sheet Payroll within the current file.

```
=GetDocument("Open Dashboard",124,,,,,TRUE,"Dashboard")
```

This example opens the target file as an Axiom form. Because a web tab name is defined, the form will open within the application instead of in the user's browser.

ManagePlanFileAttachments function

Opens a dialog that allows the user to add, edit, or delete an attachment. This function can be used in an Axiom file to provide an alternate method of managing plan file attachments.

> **NOTE:** ManagePlanFileAttachments is not for use in Axiom forms. If you want users to be able to upload attachments from an Axiom form, then you must use a Button component with the **Upload Plan File Attachment** button behavior. For more information, see the *Axiom Forms and Dashboards Guide*.

`ManagePlanFileAttachments("`*`DisplayText`*`", "`*`FileGroupName`*`", `*`PlanFileCode`*`)`

Parameter	Description
DisplayText	The text to display in the cell.
	This text displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.
	This parameter supports use of the variable `{count}`, which can be used to display the count of attachments for the plan file.
FileGroupName	The name of the file group that the plan file belongs to. A file group alias name can also be used.
PlanFileCode	The plan file code for which you want to manage attachments, such as 21000.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- Users must have read/write access to the plan file in order to manage attachments. If the user has read-only access to the plan file, then the Browse Attachments dialog opens instead, so that the user can open existing attachments (same as when using GetPlanFileAttachment). If the user has no access to the plan file, then an error message is shown.

- ManagePlanFileAttachments is a non-volatile function. This means that if the `{count}` variable is used, it will not automatically update when the count changes—the sheet must be refreshed in order to update the function.

▶ Examples

`=ManagePlanFileAttachments("Manage Attachments for Dept 21000","Budget 2019",21000)`

This example opens the Manage Attachments dialog so that the user can add, edit, or delete attachments for plan file 21000 in file group Budget 2019.

`=ManagePlanFileAttachments("Manage Attachments for Dept 21000","Current Budget",21000)`

This example is the same as the first example, except that this time a file group alias name is used. When the alias is edited to point to a different file group, the function will update as well. In this case, the plan file codes should be queried dynamically and then referenced in the function, to ensure that the function is using the correct codes for the current file group.

RunEvent function

Runs any Scheduler jobs that contain the specified event. The jobs are added to the schedule and are eligible to be processed immediately, depending on Scheduler thread availability and any other higher-priority jobs already in the queue. The processing priority for the jobs is Event Handler.

To run the jobs, the user double-clicks on the cell containing the RunEvent function. Variable values can be passed to the job using the function.

▶ Syntax

```
RunEvent("DisplayName", "EventName", "PreConfirmationMessage",
"PostConfirmationMessage", "VariableValuePair1", "VariableValuePair2",
[Additional VariableValue pairs])
```

Parameter	Description
DisplayName	The display text for the cell containing the RunEvent function. For example, the text could be something like "Click here to process plan files."
	The DisplayName displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.
EventName	The name of the event handler that you want to run. Any Scheduler jobs that contain an active event handler with this name will be run.
PreConfirmationMessage	Optional. A custom message to display to the user before the jobs are added to the schedule. The user has the option to continue the operation, or cancel.
	If no custom message is defined, then the following generic confirmation is displayed: "You are about to run the event *EventName*. Do you wish to continue?"
	If the user clicks **Yes**, the jobs are added to the schedule. If the user clicks **No**, the process is canceled and no action occurs.

Parameter	Description
PostConfirmationMessage	Optional. A custom message to display to the user after the jobs have been added to the schedule. This is for information purposes only; the only action the user can take is to dismiss the dialog.
	If no custom message is defined, then the following generic confirmation message is displayed: "Event *EventName* has been scheduled."
	NOTE: If no jobs contain an active event handler with the specified name, then the message "No jobs found for event *EventName*" is displayed. This message is displayed regardless of whether a custom message is defined.
VariableValuePairs	The function arguments can contain one or more variable/value pairs, up to 10 pairs. Each pair must be listed using the format "*VariableName=Value*" (with or without spaces around the equals sign).
	If the specified name matches a job variable used in the job, then that variable will be replaced with the specified value. If no matching variable is found in the job, then the variable/value pair is ignored.

▶ Remarks

- The user who double-clicks the RunEvent cell to trigger the job execution does not need to have security permissions for Scheduler, or to any of the jobs that contain the specified event handler. If the event handler is configured to run as Requester (instead of Owner), then the user must have the appropriate rights to any files to be processed by the job.
- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use RunEvent.
- RunEvent is a volatile function.

▶ Examples

```
RunEvent("Click here to process plan files","ProcessPlanFiles")
```

This example executes any job that contains an event handler named ProcessPlanFiles.

```
RunEvent("Click here to process plan files","ProcessPlanFiles",,,"filter =
region='north'")
```

This example executes any job that contains an event handler named ProcessPlanFiles. If the job uses a variable named {filter} to define the list of plan files to process, then the specified filter will be passed to the job, and only departments belonging to the North region will be processed.

```
RunEvent("Click here to process plan files","ProcessPlanFiles","You are about
to process plan files for the "&C3&" region. Do you want to
continue?",,"filter = "&C4)
```

This example is the same as the previous example, except that a custom pre-execution message is being displayed to the user. Additionally, cell references are used so that the message and the filter are dynamic, depending on what region is being processed.

ShowFilterWizardDialog function

Opens the Filter Wizard in advanced view so that users can create a filter. Users can double-click on the cell that contains the function to open the wizard. The filter is then saved to a designed cell.

This function is intended to be used within spreadsheet Axiom files, to launch the Filter Wizard and place a filter in a cell. The filter can then be referenced as needed by Axiom queries, sheet filters, and so on. It is up to the file designer to configure the file to use the filter in some way.

> **NOTES:**
> - The ShowFilterWizardDialog function is not for use in Axiom forms. If you want to allow users to create a filter in Axiom forms, you can use the Filter Wizard command with a button, or you can use refresh variables.
>
> - The ShowFilterWizardDialog function does not trigger a data refresh once the filter is created. If the file is designed to use the filter in Axiom queries or other data queries, the user would have to manually refresh the file afterward to apply the filter. If you want the user to define a filter as part of a data refresh, you can use refresh variables instead.

▶ Syntax

```
ShowFilterWizardDialog("DisplayText", "TargetCell", "PrimaryTableName",
"LimitColumn", "DialogTitle")
```

Parameter	Description
DisplayText	The text to display in the cell.
	The DisplayText displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.

Parameter	Description
TargetCell	The target cell in which to place the filter. You can enter any of the following: • A standard cell reference, with or without the sheet name. For example: "B2" or "Variables!B2". • A column reference, such as "H" or "AZ". The filter will be placed in that column, within the same row as the cell with the function. • A relative column reference, using the format +N or -N. The filter will be placed N columns to the right or left, within the same row as the function. For example, "+1" places the filter one column to the right of the cell with the function, and "-1" places the filter one column to the left of the cell with the function. **NOTE:** Remember to place the target cell location in quotation marks, such as "C25". If you omit the quotation marks and use just C25, then the entry is interpreted as a cell reference within the function—meaning that Axiom Software will attempt to read the target cell location from the specified cell. If C25 is empty, the function will behave as if no target cell was specified.
PrimaryTableName	Optional. Complete this parameter as follows, depending on the objective: • **Create a filter using any table**: Omit both the PrimaryTableName and the LimitColumn if you want to open the Filter Wizard dialog showing all available tables. The user can create a filter using any table. • **Create a filter based on a primary table**: Specify a PrimaryTableName if you want to open the Filter Wizard dialog and only show tables that are valid to filter the primary table. The user can create a filter based on the primary table or on a lookup reference table. This is the recommended approach if the file is designed to apply the filter to a specific data query, so that you can be sure the filter will be valid in the context of the query. • **Create a limit query statement**: Specify a PrimaryTableName and/or a LimitColumn name if you want to use the Filter Wizard to create a limit query statement for an Axiom query. See the description of the LimitColumn parameter for more information. In this context, the primary table 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.

Parameter	Description
LimitColumn	Optional. Complete this parameter if you want to use the Filter Wizard to create a limit query statement for an Axiom query. Otherwise, leave this parameter blank to create a standard filter criteria statement.
	This parameter specifies the column to limit in the Axiom query. The column listed in this parameter 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 PrimaryTableName parameter. 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 PrimaryTableName field. When the limit query statement is created, only the column name will be listed in the Limit parameter.
	For more information, see the *Axiom File Setup Guide*.
DialogTitle	Optional. Specifies custom title text for the Filter Wizard dialog. If omitted, the standard title of "Filter Wizard" is used.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- The function uses the advanced view of the Filter Wizard dialog. It is not possible for the user to switch to simple view (hierarchies) when using this function.

- The function uses the web version of the Filter Wizard dialog.

- When the user clicks OK in the Filter Wizard to complete the filter creation, the filter is placed in the designated target cell, overwriting any current contents. If the target cell already has contents when the Filter Wizard is opened by the function, these contents are displayed in the Filter box (regardless of whether the contents comprise a valid filter criteria statement).

- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use the ShowFilterWizardDialog function.

- The ShowFilterWizardDialog function is a non-volatile function.

▶ Examples (standard filter criteria statements)

`=ShowFilterWizardDialog("Filter Data","A10")`

This example allows the user to create a filter using any customer table in the system. The filter will be placed in cell A10 on the current sheet.

`=ShowFilterWizardDialog("Filter Data","H","GL2018")`

This example allows the user to create a filter that is valid for use against the GL2018 primary table. The wizard only shows the GL2018 table as well as any lookup reference tables (such as Dept and Acct). The filter will be placed in column H, in the same row as the cell containing the function.

`=ShowFilterWizardDialog("Filter Data","+2","GL2018")`

This example is the same as the previous example, except this time the filter will be placed in the column 2 cells to the right of the cell containing the function, within the current row. If the function is in cell C15, the filter would be placed in cell E15.

▶ Examples (limit query statements)

`=ShowFilterWizardDialog("Limit Query","A10",,"Dept.Dept","Limit Query")`

This example allows the user to create a limit query statement for an Axiom query, to limit the data in the Dept column based on another table. The user will first select a table to limit the query by. The user can select any table with a lookup to the limit column Dept.Dept. Note that the dialog title also uses the custom text "Limit Query", as defined in the last parameter.
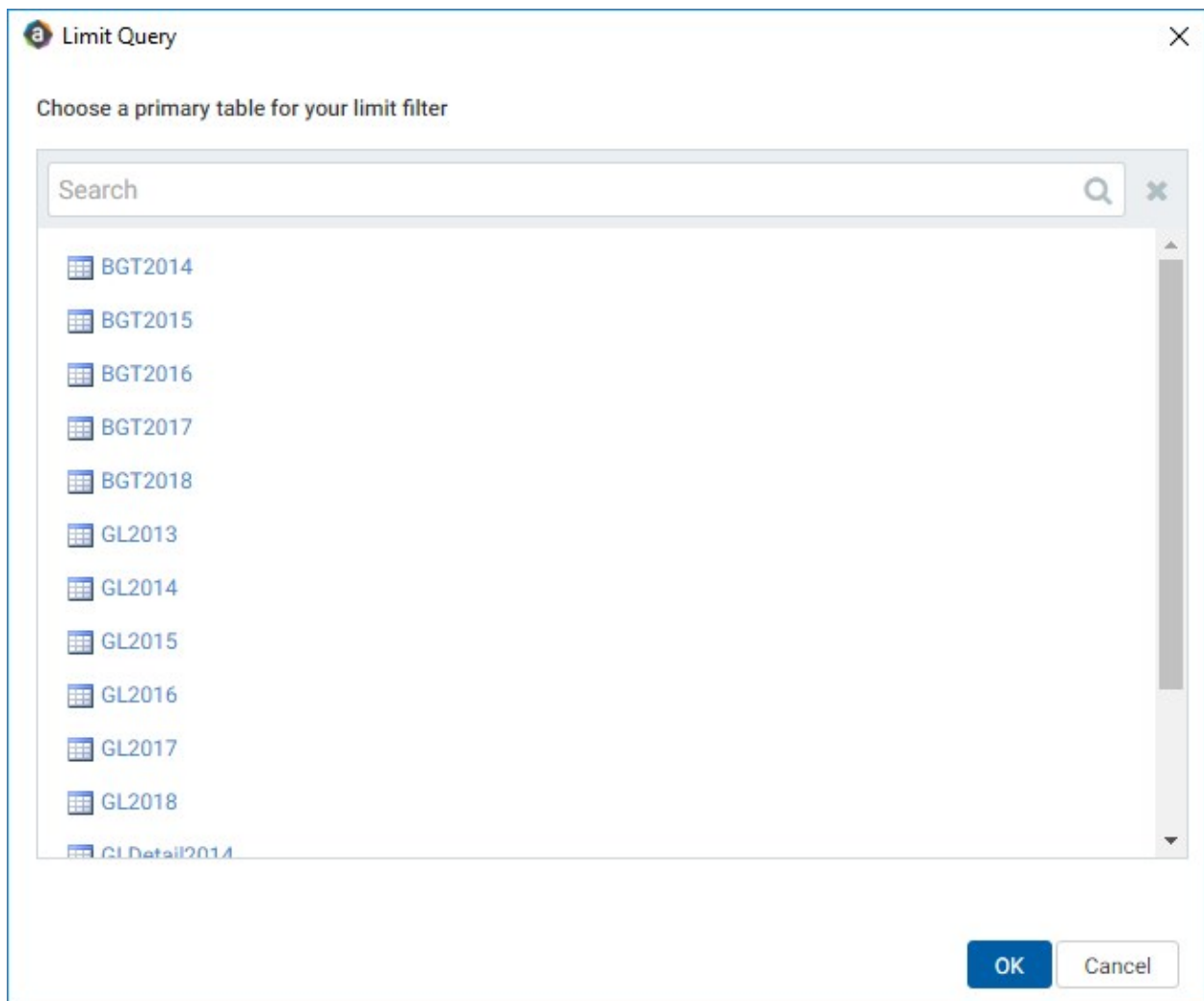
Then the user can create a filter on the selected table, to further limit the query. When the user clicks apply, the limit query statement is placed in the Filter box. The user can further modify the statement if needed, and then click OK to place the statement into cell A10 (the target cell in this example).

The limit query statement can then 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.

`=ShowFilterWizardDialog("Limit Query","A10","GL2018","Dept")`

In this example, the primary table and the limit column are listed separately. The limit column is GL2018.Dept, which looks up to Dept.Dept. When the limit query statement is created, only the column name is listed for the Limit parameter, instead of a full Table.Column name. You might use this approach if you need the limit column to be deliberately ambiguous.

ShowFormDialog function

Opens a designated form-enabled file as a form dialog. Users can double-click on the cell that contains the function to open the form.

This function is intended to be used within spreadsheet Axiom files, to launch an Axiom form that is designed to be used as a dialog. The behavior is the same as when using the Show Form Dialog command for custom task panes and ribbon tabs.

This function can be used in one of two ways:

- As a "hyperlink" to launch a dialog to perform a particular task. The user can open the dialog from the file, perform the task, and then return to the file. The actions performed in the dialog have no impact on the source file. This use case is just an alternate way for users to get to the dialog.

- As a user interface to perform an action on the current file. The user can open the dialog from the file, make selections in the dialog, and then apply those selections to the current file using form state or by using commands that can be executed on the client. In this use case, the form is specifically designed to work together with the file where it is being launched.

▶ Syntax

```
ShowFormDialog("DisplayText", "DocumentPath", "FormState")
```

Parameter	Description
DisplayText	The text to display in the cell. The DisplayText displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.
DocumentPath	The form-enabled file to open when the user double-clicks on the cell. Specify the file name and path in the Axiom Software virtual file system. The path is the folder path as it would display in Axiom Explorer—for example: `\Axiom\Reports Library\Forms\SelectionForm.xlsx`.
FormState	Optional. One or more form state key/value pairs to pass to the target file when it is opened. Form state key/value pairs are specified as follows: `FormStateKey1=Value;FormStateKey2=Value` Separate multiple form state key/value pairs using semicolons. If a value contains a semicolon, then it must be preceded by a backslash (\) so that Axiom does not treat the semicolon as a delimiter. Equals signs within a value (such as to pass a filter criteria statement as a value) do not need to be specially treated. Form state values can be read in the target file by using the function GetFormState("FormStateKey"), or by configuring a component to use the `[FormState=FormStateKey]` tag.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Passing form state values

When you use the FormState parameter to pass form state values from the current spreadsheet into the target form, the behavior is as follows:

- When the ShowFormDialog function is double-clicked, the designated form state values are set in the current spreadsheet, which causes them to be passed into the target form when it is opened. In the target form, GetFormState functions and components using the [FormState] tag for the designated keys will return the values defined in the function.

- The form state values in the ShowFormDialog function overwrite any existing values for the designated form state keys in the spreadsheet. For example, if the spreadsheet has an existing value Blue for the form state key Color, and the function contains Color=Red, then when the function is used the value of Color is set to Red. This is the only way that the spreadsheet can change an existing form state value. Form state values set by ShowFormDialog are actual form state values, not default values.

- If the spreadsheet contains GetFormState functions for the designated form state keys, these functions will not update to show the new value until the spreadsheet is refreshed. It is not necessary for the spreadsheet to contain any GetFormState functions—the values in the ShowFormDialog function can be used solely to impact the target form.

- If the spreadsheet contains other form state values that are not listed in the ShowFormDialog function, those existing values are passed into the target form as normal.

▶ Remarks

- To obtain a file path for use in the function, you can right-click a file in Axiom Explorer or the Explorer task pane and choose **Copy document path to clipboard**, then paste the path into the function parameters (or into a cell that you will reference).

- The Axiom Double-Click setting does *not* need to be enabled for the sheet in order to use ShowFormDialog.

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- ShowFormDialog is a non-volatile function.

▶ Examples

```
=ShowFormDialog("Select Items","\Axiom\Reports
Library\Forms\SelectionForm.xlsx")
```

This example opens the specified form-enabled file as a dialog.

```
=ShowFormDialog("View Details","\Axiom\File Groups\Budget
2019\Utilities\Details.xlsx";"Acct=6000")
```

This example opens the specified form-enabled file as a dialog, and sets the value of the Acct form state key to 6000.

Axiom Form Functions

This section contains information on Axiom functions that are intended for use with Axiom forms. These functions can be used to enable navigation between forms and to share information between forms.

GetFormDocumentLinkTag function

Generates a hyperlink tag to an Axiom form in the Axiom file system, using the special content tags for use in Formatted Grid components. When the source file containing the content tag is viewed as an Axiom form, the tag will render as a hyperlink to the specified file.

This function is only for use within form-enabled files. The content tags generated by the function have no effect within the source file; the file must be viewed as a form in order to use the hyperlink.

▶ Syntax

```
GetFormDocumentLinkTag("DisplayText", DocIDorPath, "SheetFilter",
OpenInNewWindow, GeneratePDF, "Variables")
```

Parameter	Description
DisplayText	The display text for the hyperlink.
DocIDorPath	The file for which you want to create a hyperlink. You can specify this file by entering either of the following: • The document ID of the file, such as 93. • The full path to the file, such as `\Axiom\Reports Library\Forms\Dashboard.xlsx`.
SheetFilter	Optional. A filter to be applied as a temporary sheet filter to the target file. This parameter specifies both the type of filter (table type, table, etc.) and the filter criteria statement. See the following section for more information on the sheet filter syntax. When the target file is opened via hyperlink, the specified filter is applied as a sheet filter behind the scenes. The filter does not display on the Control Sheet, and will not be saved in the file (similar to Quick Filter functionality).

Parameter	Description
OpenInNewWindow	Optional. A Boolean value that specifies whether the Axiom form is opened in a new window or the current window: • `FALSE` (Default): The document will be opened in the current window (or tab), replacing the current contents. • `TRUE`: The document will be opened in a new window (or tab).
GeneratePDF	Optional. A Boolean value that specifies whether the hyperlink will generate a PDF copy of the form: • `FALSE` (Default): The form will be opened normally. • `TRUE`: A PDF will be generated for the form and this PDF will be opened instead of the form. This parameter is intended to be used to enable PDF printing of an Axiom form.
Variables	Optional. One or more variable / value pairs to pass to the target file when it is opened. Variable / value pairs are specified as follows: *VariableName=Value*;*VariableName=Value* Separate multiple variable / value pairs using semicolons. If a variable value contains a semicolon, then it must be preceded by a backslash (\) so that Axiom does not treat the semicolon as a delimiter. Equals signs within a value (such as to pass a filter criteria statement as a value) do not need to be specially treated. Variable values can be read in the target file by using the GetDocumentInfo function. For more information and examples, see Passing values from one file to another.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Sheet filter syntax

The sheet filter parameter uses the following syntax:

```
FilterTarget;CriteriaStatement
```

The *FilterTarget* is a table name or a table type name. It specifies the tables to be affected by the sheet filter.

- Reference table: If the filter target is a reference table (such as DEPT), then the filter is applied to all tables that link to the DEPT table.
- Table type: If the filter target is a table type (such as GL), then the filter is applied to all tables in the table type.

- Data table: If the filter target is a data table (such as GL2018), and the table belongs to a table type, then the filter is applied to all tables in the table type. Otherwise, the filter is applied to the specified table.

> **NOTE:** Axiom Software checks for matches in the order listed above. Therefore if a reference table and a table type share the same name, the target will be the reference table.

The *CriteriaStatement* is the filter criteria statement to apply to the affected tables. Standard filter criteria syntax applied. For example:

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

This filter applies to all tables that link to the DEPT table, and filters the data for the North region.

> **NOTES:**
> - When using the sheet filter parameter, keep in mind that the temporary filter will be concatenated with any existing sheet filters in the file using AND. If the document has existing sheet filters, you should test the GetDocumentHyperlink filter to ensure that the combination of filters returns data as you intended.
> - The function GetCurrentValue can be used in the target file to return information about the temporary sheet filter.

▶ Remarks

- Both GetFormDocumentLinkTag and GetFormDocumentURL can be used to generate a hyperlink to an Axiom form within a Formatted Grid component. GetFormDocumentLinkTag generates a complete but basic set of HREF content tags that cannot be further modified. GetFormDocumentURL generates a URL that can be used within a set of HREF content tags that you create manually, thereby allowing more flexibility in how the content tags are set up.

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- GetFormDocumentLinkTag is a non-volatile function.

▶ Examples

```
=GetFormDocumentLinkTag("Open KPI dashboard", "\Axiom\Reports
Library\Dashboards\kpi_dashboard.xlsx")
```

This example creates a content tag that will render as a hyperlink to the specified document, and open it as an Axiom form. Although you can specify the file path (or document ID) for the second parameter directly, in many cases you will use this function in conjunction with the GetPlanFilePath function, to look up the appropriate file path for a plan file. For example:

```
=GetFormDocumentLinkTag("Open plan file for Dept 3000", GetPlanFilePath
("Budget 2019",3000), , TRUE)
```

This example creates a content tag that will render as a hyperlink to the plan file for department 3000 in file group Budget 2019, and open it as an Axiom form. In the spreadsheet, the resulting tag will look something like the following:

```
[href=http://servername/Axiom/forms/psMRBzzjVcs0yAjqyOAyZQ__; Text=Open
plan file for Dept 3000; UseNewWindow=true]
```

In the rendered Axiom form, the resulting hyperlink will look something like the following (assuming the appropriate cell formatting):

[Open plan file for Dept 3000](#)

```
=GetFormDocumentLinkTag("Open KPI dashboard", 93, "GL;Dept.Region='North'")
```

This example is the same as the first example, except:

- A document ID is used instead of a file path to identify the file.
- A filter is applied to limit the data in the Axiom form to data from the North region.

```
=GetFormDocumentLinkTag("Print plan file", GetPlanFilePath("Budget 2019",
3000), , TRUE, TRUE)
```

In this example, the GeneratePDF parameter is used to automatically generate a PDF for printing. The content tags that result from this function will automatically append /pdf to the end of the URL, such as:

```
[href=http://servername/Axiom/forms/psMRBzzjVcs0yAjqyOAyZQ__/pdf;
Text=Print plan file; UseNewWindow=true]
```

```
=GetFormDocumentLinkTag("See Detail", 93, , True, ,
"Region=West;Category=Payroll")
```

In this example, values for the variables Region and Category are being sent to the target form. If the target form uses GetDocumentInfo functions to return the variable values, the values West and Payroll will be returned and can be used to impact data queries in the file (or for some other purpose). Additionally, the OpenInNewWindow parameter is used to open the target form in a new window.

GetFormDocumentURL function

Generates a URL to a form-enabled file within the Axiom file system, to open the file as an Axiom form instead of as a spreadsheet. If this URL is used in content tags within a Formatted Grid component, it will result in a clickable hyperlink to open the file from within an Axiom form.

This function is specifically intended for situations where you are linking to other Axiom forms from within an Axiom form.

▶ Syntax

```
GetFormDocumentURL(DocIDorPath, "SheetFilter", "Variables")
```

Parameter	Description
DocIDorPath	The file for which you want to return a URL. You can specify this file by entering either of the following: • The document ID of the file, such as 93. • The full path to the file, such as `\Axiom\Reports Library\Forms\Dashboard.xlsx`.
SheetFilter	Optional. A filter to be applied as a temporary sheet filter to the target file. This parameter specifies both the type of filter (table type, table, etc.) and the filter criteria statement. See the following section for more information on the sheet filter syntax. When the target file is opened via hyperlink, the specified filter is applied as a sheet filter behind the scenes. The filter does not display on the Control Sheet, and will not be saved in the file (similar to Quick Filter functionality).
Variables	Optional. One or more variable / value pairs to pass to the target file when it is opened. Variable / value pairs are specified as follows: *VariableName=Value*;*VariableName=Value* Separate multiple variable / value pairs using semicolons. If a variable value contains a semicolon, then it must be preceded by a backslash (\) so that Axiom does not treat the semicolon as a delimiter. Equals signs within a value (such as to pass a filter criteria statement as a value) do not need to be specially treated. Variable values can be read in the target file by using the GetDocumentInfo function. For more information and examples, see Passing values from one file to another.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Sheet filter syntax

The sheet filter parameter uses the following syntax:

```
FilterTarget;CriteriaStatement
```

The *FilterTarget* is a table name or a table type name. It specifies the tables to be affected by the sheet filter.

- Reference table: If the filter target is a reference table (such as DEPT), then the filter is applied to all tables that link to the DEPT table.
- Table type: If the filter target is a table type (such as GL), then the filter is applied to all tables in the table type.

- Data table: If the filter target is a data table (such as GL2018), and the table belongs to a table type, then the filter is applied to all tables in the table type. Otherwise, the filter is applied to the specified table.

> **NOTE:** Axiom Software checks for matches in the order listed above. Therefore if a reference table and a table type share the same name, the target will be the reference table.

The *CriteriaStatement* is the filter criteria statement to apply to the affected tables. Standard filter criteria syntax applied. For example:

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

This filter applies to all tables that link to the DEPT table, and filters the data for the North region.

> **NOTES:**
> - When using the sheet filter parameter, keep in mind that the temporary filter will be concatenated with any existing sheet filters in the file using AND. If the document has existing sheet filters, you should test the GetDocumentHyperlink filter to ensure that the combination of filters returns data as you intended.
> - The function GetCurrentValue can be used in the target file to return information about the temporary sheet filter.

▶ Remarks

- Both GetFormDocumentLinkTag and GetFormDocumentURL can be used to generate a hyperlink to an Axiom form within a Formatted Grid component. GetFormDocumentLinkTag generates a complete but basic set of HREF content tags that cannot be further modified. GetFormDocumentURL generates a URL that can be used within a set of HREF content tags that you create manually, thereby allowing more flexibility in how the content tags are set up.

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- GetFormDocumentURL is a non-volatile function.

▶ Examples

```
=GetFormDocumentURL(930)
```

This example generates a URL to an Axiom form within the Axiom file system. If used on its own like this, you could copy the URL into a Hyperlink component. However in most cases you would use the function as part of content tags within a Formatted Grid component. For example:

```
="[Href="&GetFormDocumentURL(G23)&";Text=Open
dashboard;UseNewWindow=True]"
```

In this case, the GetFormDocumentURL function is used within a formula to create a full string of content tags. The function is looking up the file path or document ID from cell G23.

The content tags do not have any effect within the spreadsheet itself. In order to generate a hyperlink, the content tags would be used within a cell that is part of a data source for a Formatted Grid component. When the source file is viewed as an Axiom form, the tags would be rendered as a clickable hyperlink within the grid.

```
=GetFormDocumentURL("\Axiom\Reports Library\Forms\Dashboard.xlsx",
"Dept;Dept.Region='US West'")
```

This example is the same as the first example, except in this case a filter is applied to limit the data in the Axiom form when it is opened via URL. Also, a full file path is used instead of a document ID to identify the file.

```
=GetFormDocumentURL(930, , "Region=West;Category=Payroll")
```

In this example, values for the variables Region and Category are being sent to the target form. If the target form uses GetDocumentInfo functions to return the variable values, the values West and Payroll will be returned and can be used to impact data queries in the file (or for some other purpose).

GetFormResourceLinkTag function

Generates a hyperlink tag to a non-Axiom file in the Axiom file system, using the special content tags for use in Formatted Grid components. When the source file containing the content tag is viewed as an Axiom form, the tag will render as a hyperlink to the specified file.

This function is specifically intended for situations where you are using an Axiom query to bring in a list of plan file attachments, and you want to generate hyperlinks to those files so that the user can open them from the Axiom form. You would then display the hyperlinks in the form using a Formatted Grid component.

This function is only for use within form-enabled files. The content tags generated by the function have no effect within the source file; the file must be viewed as a form in order to use the hyperlink.

> **NOTE:** This function creates a hyperlink that opens the specified file in its native format—for example, in Word for DOCX files. Therefore although the target file must reside within the Axiom file system, this function should not be used to link to Axiom files such as reports, because the file would be opened in a regular Excel instance instead of within Axiom Software. This function should only be used to link to non-Axiom files.

▶ Syntax

```
GetFormResourceLinkTag("DisplayText", DocIDorPath)
```

Parameter	Description
DisplayText	The display text for the hyperlink. The appearance of the hyperlink depends on the formatting of the cell. If you want the hyperlink to appear in blue font with an underline, you must format the cell that way.
DocIDorPath	The file for which you want to create a hyperlink. You can specify this file by entering either of the following: • The document ID of the file, such as 93. • The full path to the file, such as `\Axiom\File Groups\Budget 2019\Plan File Attachments\24000\Proposal.pdf`.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- Both GetFormResourceLinkTag and GetFormResourceURL can be used to generate a hyperlink to a plan file attachment within a Formatted Grid component. GetFormResourceLinkTag generates a complete but basic set of Href content tags that cannot be further modified. GetFormResourceURL generates a URL that can be used within a set of Href content tags that you create manually, thereby allowing more flexibility in how the content tags are set up.

- The hyperlinked document always opens in a new window.

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- GetFormResourceLinkTag is a non-volatile function.

▶ Examples

```
=GetFormResourceLinkTag("Open file",930)
```

This example creates an HTML tag that will render as a hyperlink to document ID 930 when the file is viewed as an Axiom form. In the spreadsheet, the resulting HTML will look something like the following:

```
[href=http://servername/Axiom/resource/psMRBzzjVcs0yAjqyOAyZQ__;
Text=Open file]
```

In the rendered Axiom form, the resulting hyperlink will look something like the following (assuming the appropriate cell formatting):

[Open file](#)

When using the function within an Axiom query to automatically generate links, the function might be structured as follows:

```
=GetFormResourceLinkTag("Open "&F4,S4)
```

Where column F contains the names of the documents and column S contains the corresponding document IDs or file paths.

GetFormResourceURL function

Generates a URL to a non-Axiom file within the Axiom file system, to open the file in its native application. If this URL is used in content tags within a Formatted Grid component, it will result in a clickable hyperlink to open the file from an Axiom form.

This function is specifically intended for situations where you are using an Axiom query to bring in a list of plan file attachments, and you want to generate hyperlinks to those files so that the user can open them from the Axiom form. You would then display the hyperlinks in the form using a Formatted Grid component.

▶ Syntax

```
GetFormResourceURL(DocIDorPath)
```

Parameter	Description
DocIDorPath	The file for which you want to return a URL. You can specify this file by entering either of the following: • The document ID of the file, such as 93. • The full path to the file, such as `\Axiom\File Groups\Budget 2019\Plan File Attachments\24000\Proposal.pdf`.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- Both GetFormResourceLinkTag and GetFormResourceURL can be used to generate a hyperlink to a plan file attachment within a Formatted Grid component. GetFormResourceLinkTag generates a complete but basic set of Href content tags that cannot be further modified. GetFormResourceURL generates a URL that can be used within a set of Href content tags that you create manually, thereby allowing more flexibility in how the content tags are set up.

- If your system is deployed in an environment where users work in different system languages,

then you should use system folder names in any file paths.

- GetFormResourceURL is a non-volatile function.

▶ Examples

`=GetFormResourceURL(930)`

This example generates a URL to a non-Axiom file within the Axiom file system. However, the function would rarely be used on its own; instead it would be part of content tags used within a Formatted Grid component. For example:

`="[Href="&GetFormResourceURL(G23)&";Text=Open attachment]"`

In this case, the GetFormResourceURL function is used within a formula to create a full string of content tags. The function is looking up the document ID from cell G23.

The content tags do not have any effect within the spreadsheet itself. In order to generate a hyperlink, the content tags would be used within a cell that is part of a data source for a Formatted Grid component. When the source file is viewed as an Axiom form, the tags would be rendered as a clickable hyperlink within the grid.

GetFormState function

Returns the value of a given form state key that is held in form state memory for the current file.

This function is only for use in configurations where an Axiom form is being used as a dialog or task pane in the Excel Client or Windows Client, and you want to pass form state values from the Axiom form to the active spreadsheet file.

▶ Syntax

`GetFormState("FormStateKey", "DefaultValue")`

Parameter	Description
FormStateKey	The key name of a particular form state value that you want to return. For example, if an Axiom form component has been configured with `[FormState=VPName]`, then the form state key is VPName.

Parameter	Description
DefaultValue	The text to be used as the default form state value for the designated key if no value has yet been set by Apply Form State.
	Technically, this does not set the actual form state value to the default value. You can change the default value within the function, and the function will continue to return the latest default value *until an actual value is set by a form*. Once the actual form state value has been set, then the default value is ignored and the function will not respond to any future changes to this parameter during the current file session. When the file is closed, the actual form state values are cleared and the default values will again be used the next time the file is opened.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- When an Axiom form dialog or task pane is opened, all form state values from the currently active spreadsheet (whether default values or actual values) are passed into the Axiom form. These values override any default values set in the Axiom form.

- When using the GetFormState function to set a default value, make sure that the file only contains one function with a default value for each form state key. If multiple default values are set for a single form state key within a file, then the default value will be taken from the last function that happens to be evaluated when the sheet is calculated.

- GetFormState is a volatile function.

▶ Examples

`=GetFormState("VPName")`

This example returns the value of the form state key VPName—for example: Jones. If no value has yet been set for this form state key, the function will return blank.

`=GetFormState("VPName","Smith")`

This example is the same as the first example, except that if no value has yet been set for VPName, then the default value will be set to Smith and the function will return Smith. If a form subsequently sets the actual value to Jones, then the function will return Jones and the default value is ignored.

GetSharedVariable function

Returns the value of a named shared variable.

This function is only for use in composite forms, where there is a parent form with one or more child forms displayed using the Embedded Form component. Shared variables provide a way to pass information between the parent form and the child forms.

▶ Syntax

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

Parameter	Description
VariableName	The name of a particular shared variable for which you want to return the value.
DefaultValue	Optional. The value to be used for the shared variable, if no value has yet been set for the variable. When the GetSharedVariable function is calculated, Axiom Software checks the shared variable list that is stored in memory for the shared form instance. - If no value currently exists for the specified variable, the variable value is set to the default value defined in the function. This becomes the actual defined value for the variable, to be shared with the other forms in the shared form instance. - If a value already exists for the specified variable, then the default value is ignored and the function returns the defined value for the variable.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Shared variable values

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

- **SetSharedVariable function:** Each time this function is evaluated, it sets the current value of a named shared variable to the value defined in the function, overwriting any existing value for the variable. This method takes precedence over any other method of setting the variable value.

- **Interactive component:** An interactive form component, such as a combo box, can be configured to store its value using a named shared variable. Each time the component value is submitted to the source file, it sets the current value of the shared variable to the component value, overwriting any existing value for the variable.

- **Apply Shared Variable command**: A button in an Axiom form can be configured to use this command, which sets the value for one or more named shared variables. When a user clicks the button, the variables are set to the specified values, overwriting any existing value for the variables.

- **GetSharedVariable function:** If no value currently exists for a named shared variable when this function is evaluated, then the value of the variable is set to the value defined in the DefaultValue parameter of the function. This approach only works once, when the variable has no defined value. Once the variable has a value, the DefaultValue parameter is no longer evaluated and cannot affect the value of the variable.

Variable values can be set within the parent form or within any child form. Once the variable value is set, it is then available to all forms within the shared form instance, though forms must be updated in order to reflect a changed variable value.

▶ Remarks

- When an Axiom form with an Embedded Form component is first opened, the functions in the parent form are evaluated before the embedded child form. Therefore, the DefaultValue parameter in the child form only applies if the parent form does not already set a value for the shared variable.

- It is typically not feasible to use an Axiom function or a data lookup to determine the value to be used in the DefaultValue parameter, because the GetSharedVariable function will calculate and set the variable value before the other queries resolve.

- Generally speaking, shared variables are stored in memory as strings. If you want the return value to be presented as a number or date, cell formatting and/or additional conversion functions may be necessary.

- GetSharedVariable is a volatile function.

▶ Examples

`=GetSharedVariable("PlanCode")`

This example returns the value of the shared variable PlanCode—for example: 42000. If no value has yet been set for this shared variable, the function will return blank.

`=GetSharedVariable("PlanCode","1000")`

This example is the same as the first example, except that if no value has yet been set for the shared variable PlanCode, then the value will be set to 1000 and the function will return 1000. If the variable value is subsequently changed by using a SetSharedVariable function or an interactive component, then the default value of 1000 will be ignored and the function will return the currently defined value for the function.

SetSharedVariable function

Sets the value of a named shared variable, and returns the variable value into the cell.

This function is only for use in composite Axiom form configurations, where there is a parent form with one or more child forms displayed using the Embedded Form component. Shared variables provide a way to pass information between the parent form and the child forms.

▶ Syntax

SetSharedVariable("*VariableName*", "*Value*")

Parameter	Description
VariableName	The name of a particular shared variable for which you want to set the value.
Value	The value to be used for the shared variable.
	Whenever the SetSharedVariable function is evaluated, the value specified in the Value parameter is set as the variable value, overwriting any existing value for the variable.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Shared variable values

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

- **SetSharedVariable function:** Each time this function is evaluated, it sets the current value of a named shared variable to the value defined in the function, overwriting any existing value for the variable. This method takes precedence over any other method of setting the variable value.

- **Interactive component:** An interactive form component, such as a combo box, can be configured to store its value using a named shared variable. Each time the component value is submitted to the source file, it sets the current value of the shared variable to the component value, overwriting any existing value for the variable.

- **Apply Shared Variable command**: A button in an Axiom form can be configured to use this command, which sets the value for one or more named shared variables. When a user clicks the button, the variables are set to the specified values, overwriting any existing value for the variables.

- **GetSharedVariable function:** If no value currently exists for a named shared variable when this function is evaluated, then the value of the variable is set to the value defined in the DefaultValue parameter of the function. This approach only works once, when the variable has no defined value. Once the variable has a value, the DefaultValue parameter is no longer evaluated and cannot affect the value of the variable.

Variable values can be set within the parent form or within any embedded child form. Once the variable value is set, it is then available to all forms within the shared form context, though the form must be updated in order to reflect a changed variable value.

- Only one form in the shared form instance should contain a SetSharedVariable function for a given variable name. For example, the parent form can contain a SetSharedVariable function and the child forms can contain a GetSharedVariable function, or vice versa. If more than one form contains a SetSharedVariable function for the same variable name, then the variable value will not be shared between those forms because each form will set its own value as it is updated (and from the user perspective, the last form updated will win).

- Generally speaking, shared variables are stored in memory as strings. If you want the return value to be presented as a number or date, cell formatting and/or additional conversion functions may be necessary.

- SetSharedVariable is a volatile function.

▶ Examples

`=SetSharedVariable("PlanCode","42000")`

This example sets the value of the shared variable PlanCode to 42000 (and the function also returns this value). In real-life use cases, the value 42000 would most likely be returned using a data lookup query, and then that return cell would be referenced in the function, such as: `=SetSharedVariable("PlanCode",Variables!D24)`

This example would be used in the parent form to identify the current plan code for the shared form context. The child forms displayed in the Embedded Form component would then use the GetSharedVariable function to reference this value.

GetWebReportDocumentURL function

Generates a URL to a web report within the Axiom file system. This URL can then be referenced in Axiom forms or other web reports, using various features such as Hyperlink components and HREF tags in Formatted Grid components. You can also send the URL to another Axiom Software user, so that they can open the software directly to that report.

▶ Syntax

`GetFormDocumentURL("DocumentPath")`

Parameter	Description
DocumentPath	The document for which you want to return a URL. You can specify this file by entering either of the following:
	• The full path to the file, such as `\Axiom\Reports Library\Web\Dashboard.xlsx`.
	• The document ID of the file, such as 93. In this case you do not need to place the entry in double quotation marks.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- GetWebReportDocumentURL is a non-volatile function.

▶ Examples

```
=GetWebReportDocumentURL("\Axiom\Reports Library\Web\Actuals.xlsx")
```

This example generates a URL to a web report within the Axiom file system. If used on its own like this, you could copy the URL into a Hyperlink component. However you can also use the function to supply a URL to the HREF content tag for a Formatted Grid component.

```
=GetWebReportDocumentURL(8965)
```

This example is the same as the first example, except in this case a document ID to identify the web report.

Data Retrieval Functions

This section contains information on Axiom functions that return data—from data held in customer tables to system data.

GetData function

The GetData function queries the database and returns data from a table, given a column name, a filter criteria statement, and a table name. The GetData function can be used to return any type of data.

> **IMPORTANT:** If it is possible to use an Axiom query or a data lookup instead of GetData functions to return a set of data, it is always recommended to do so for faster performance. Use of GetData functions in a file can significantly impact the performance of that file, especially when large numbers of functions are used.

▶ Syntax

```
GetData("ColumnName", "FilterCriteria", "TableName", "NoDataDefaultMessage",
IgnoreSheetFilter, "InvalidQueryMessage", "AlternateAggregation",
"ConversionTarget")
```

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.
	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.

Parameter	Description
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.
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 table name can be omitted.
NoDataDefaultMessage	Optional. A custom message to use as the return value if the query does not return any data.
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.

Parameter	Description
AlternateAggregation	Optional. Specifies an alternate aggregation type for the returned data: • `Count`: Returns the count of records in the query set. • `DistinctCount`: Returns the count of unique records in the query set. For example, if the query returns two records that both have a value of 10, Count will return 2 but DistinctCount will return 1. • `Min`: Returns the minimum record in the query set. • `Max`: Returns the maximum record in the query set. • `Sum`: Returns the sum of records in the query set. Does not apply when querying string or date columns. • `Avg`: Returns the average value of records in the query set. Does not apply when querying string or date columns. The default aggregation behavior is the same as for Axiom queries. If the filter criteria statement for the query results in a single record, then aggregation does not apply. **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.
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).

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ 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.

- The GetData Function Wizard is available in Axiom files to assist you in building GetData functions. Right-click in any cell and then select **Axiom Wizards > GetData Function Wizard**.

- The GetData function is a non-volatile function. Use caution when linking GetData functions to volatile functions (for example, referencing a cell that contains a volatile function) as this will make the GetData function behave like a volatile function, which may impact performance and cause unexpected calculation behavior. In particular, functions that return a different result on every calculation (such as NOW) should not be used within GetData functions.

- 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.

▶ Common examples

`=GetData("M1","ACCT.AcctGroup='Benefits'","GL2018")`

This example returns the sum of all data for benefit accounts in column M1 of the GL2018 table. You could return the same result by using a fully qualified column name in the first parameter and omitting the table name in the third parameter: `=GetData("GL2018.M1","ACCT.AcctGroup='Benefits'")`

`=GetData("CYA1","ACCT.AcctGroup='Benefits'")`

This example returns the same data as the first example, assuming that CYA1 is an alias name for `GL2018.M1`. Since CYA1 is an alias name, the table name can be omitted. However, you could also write this function as `=GetData("CYA1","ACCT.AcctGroup='Benefits'","Alias")` or `=GetData("CYA1","ACCT.AcctGroup='Benefits'","GL2018")`

`=GetData("M1",,"GL2018")`

This example returns the sum of all data in M1. This example could also be written as: `=GetData("GL2018.M1")` or `=GetData("CYA1")`

```
=GetData("Description","ACCT=1000","ACCT")
```

This example queries a reference table to return a single value—in this case, the description for Acct 1000 (for example, "Cash"). This function could also be written as: `=GetData("ACCT.Description","ACCT=1000")`

▶ Optional parameter examples

```
=GetData("CYA1","ACCT.AcctGroup='Benefits'",,"Query returned no
data",,"Invalid query")
```

This example defines custom messages to display to the user if the query returns no data or if the query is invalid. Note that the unused third and fifth parameters are delimited with commas.

```
=GetData("CYA1","ACCT.AcctGroup='Benefits'",,"Query returned no
data",True,"Invalid query")
```

This example ignores any sheet filter defined on the Control Sheet. For example, if the sheet filter were `DEPT=2000`, then this function would ignore the filter and return the data for all departments.

```
=GetData("DEPT",,"DEPT",,,,"Count")
```

This example uses the alternate aggregation to return the count of all departments in the DEPT table. Keep in mind that Count will always return the full number of records in the query, regardless of which column is specified. For example, `GetData("Region",,"DEPT",,,,"Count")` will return the same number as the example above. If you need the number of unique regions in the query, then you should use DistinctCount instead: `GetData("Region",,"DEPT",,,,"DistinctCount")`

```
=GetData("M1","ACCT.AcctGroup='Benefits'","GL2018",,,,,"GBP")
```

This example uses the conversion target to return converted data. Instead of returning data "as is" from the M1 column, the data will be converted per the conversion configuration set up for this table. In this example, the data is being converted from its "base" currency (say, U.S. dollars) to British pounds.

▶ System table example

```
=GetData("TableName","Alias='CYA1'","Axiom.Aliases")
```

This example queries the system table Axiom.Aliases. It will return the name of the table that the CYA1 alias is currently assigned to—for example, GL2018.

GetFeatureInfo function

Returns information about a feature installed by a product package. This function only returns values in systems where a standardized product package has been installed for use in the Axiom Software platform.

This function is primarily intended for use by product designers, so that feature information can be referenced in feature components, and so that certain functionality can be dynamically enabled or disabled depending on the presence of certain features.

```
GetFeatureInfo("Code", "Feature")
```

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. If you do not know the feature name, you can find it on the Axiom Software Launch Page in the Web Client, under Installed Products. This section only displays if a packaged product has been installed to your system.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

► Remarks

GetFeatureInfo is a non-volatile function.

► Examples

```
=GetFeatureInfo("Version","Capital Planning")
```

This example returns the version number of the installed feature named Capital Planning, such as 01.05.00.

```
=GetFeatureInfo("DateApplied","Capital Planning")
```

This example returns the date and time that the Capital Planning feature was last installed, such as 4/7/2015 7:58:38 PM.

GetPeriod function

Returns the current period for the system or for a specified table.

```
GetPeriod("PeriodSource", "TableName")
```

Parameter	Description
PeriodSource	Specify which current period you want to return: • `System`: Returns the current period for the system, as defined in **Administration > Tables > Table Management > System Current Period / Year**. • `Table`: Returns the current period for the specified table, as defined for the table in **Administration > Tables > Table Management > Table Current Periods**. • `SystemYear`: Returns the current year for the system, as defined in **Administration > Tables > Table Management > System Current Period / Year**.
TableName	If the PeriodSource is `Table`, specify the table name.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetPeriod is a non-volatile function.

▶ Examples

```
=GetPeriod("System")
```

This example returns the system current period; for example: 6.

```
=GetPeriod("Table","Pay2018")
```

This example returns the current period for the Pay2018 table; for example: 20.

```
=GetPeriod("SystemYear")
```

This example returns the system current year; for example: 2018.

GetProcessInfo function

The GetProcessInfo function returns information on a process in Process Management, such as the currently active step in the process, or the owner of a particular step. The function has different syntax depending on the keyword used in the first parameter, which determines the type of process information to be returned.

You can use the GetProcessInfo function to return the following information:

Item	See
Current step	GetProcessInfo("CurrentStep")
Process definition ID	GetProcessInfo("ProcessDefinitionID")
Process initiator	GetProcessInfo("ProcessInitiator")
Process name	GetProcessInfo("ProcessName")
Step completed by	GetProcessInfo("StepCompletedBy")
Step completed date	GetProcessInfo("StepCompletedDate")
Step due date	GetProcessInfo("StepDueDate")
Step name	GetProcessInfo("StepName")
Step owner	GetProcessInfo("StepOwner")
Step start date	GetProcessInfo("StepStartDate")
Step status	GetProcessInfo("StepStatus")
Step status details	GetProcessInfo("StepStatusDetails")
Step type	GetProcessInfo("StepType")
Time in step	GetProcessInfo("TimeinStep")

GetProcessInfo("CurrentStep")

If "CurrentStep" is specified as the first parameter, GetProcessInfo returns the current step number of an active process.

▶ Syntax

GetProcessInfo("CurrentStep", *ProcessDefinitionID*, *PlanCode*)

Parameter	Description
CurrentStep	Use the keyword `CurrentStep` to return the current step number for a process.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file process definitions, where all steps are associated with a plan code. Cannot be used for plan file subprocesses.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

► Remarks

- If the process is not currently active, GetProcessInfo returns the text "Not Started." If you are reporting against a historical instance of the process, GetProcessInfo returns the test "Not Active."

- If the active step of a process is a subprocess, then GetProcessInfo returns the number of the parent step. For example, if step 2 is a parallel subprocess with 3 sub-steps, then when step 2 is active GetProcessInfo will only return step 2. It is not possible to return the active sub-step using GetProcessInfo.

- GetProcessInfo is a non-volatile function.

► Examples

`=GetProcessInfo("CurrentStep",5589)`

This example returns the current step of the designated process. For example: 5.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("CurrentStep",F25,42000)`

In this example, the PlanCode parameter is used to return the current step for plan file 42000. This approach is only supported when the process is a plan file process definition. Also, a cell reference is used for the process definition ID.

GetProcessInfo("ProcessDefinitionID")

If "ProcessDefinitionID" is specified as the first parameter, GetProcessInfo returns the ID of a process definition. This ID can then be used in other versions of the GetProcessInfo function to return information about that process.

► Syntax

`GetProcessInfo("ProcessDefinitionID", "ProcessDefinitionNameorPath", "FileGroupName")`

Parameter	Description
ProcessDefinitionID	Use the keyword `ProcessDefinitionID` to return the ID of a process definition.

Parameter	Description
ProcessDefinitionName orPath	The process definition for which to return the ID. Enter one of the following: - If the process definition belongs to a file group, you can enter the name of the process definition file (without the file extension). You can also optionally enter the full path to the process definition (including the file extension). The display name cannot be used. - If the process definition is located in the Process Definition Library, enter the full path to the process definition (including the file extension).
FileGroupName	Optional. The name of the file group that the process belongs to. A file group alias name can also be used. If the function is used in a file that belongs to a file group, then this parameter can be omitted and the current file group is assumed. (However, in this case you should use GetFileGroupProperty ("ProcessDefinitionID") instead.) You can also omit the name of the file group if you specified the full path to the process definition in the second parameter.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If a process has historical instances and you want to report against that history, you can look up the Process Definition ID for historical instances using **Manage > Process Management > Current Processes**. Select the check box for **Show inactive processes** (if necessary), then select the current process definition in the top of the dialog. You can then review the process history at the bottom of the dialog to find the Process Definition ID for a historical instance. This historical ID can be used in other GetProcessInfo functions.

- GetProcessInfo is a volatile function.

▶ Examples

```
=GetProcessInfo("ProcessDefinitionID","\Axiom\Process Definition
Library\Rollover.axp")
```

This example returns the ID of a process definition that is stored in the Process Definition Library, as indicated by the full path used in the second parameter. The third parameter is not needed and is omitted.

```
=GetProcessInfo("ProcessDefinitionID","Budget Process","Budget 2019")
```

This example is for a file group-specific process, so the second parameter uses just the process name. The third parameter specifies the file group that the process belongs to. The file group name could be omitted when using this function in a file that belongs to a file group.

Alternatively, you could use the full path to the file group process in the second parameter, and then the file group name would not be necessary in any context.

GetProcessInfo("ProcessInitiator")

If "ProcessInitiator" is specified as the first parameter, GetProcessInfo returns the full name of the user that started the process.

For plan file processes, the return value is the name of the user who started the specified plan file in the process. This is typically used with processes for on-demand file groups.

▶ Syntax

```
GetProcessInfo("ProcessInitiator", ProcessDefinitionID, PlanCode)
```

Parameter	Description
ProcessInitiator	Use the keyword `ProcessInitiator` to return the user who started the process.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file process definitions, where all steps are associated with a plan code. Cannot be used for plan file subprocesses.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetProcessInfo is a non-volatile function.

▶ Examples

```
=GetProcessInfo("ProcessInitiator",5589)
```

This example returns the initiator of the designated process. For example: Jane Doe.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

```
=GetProcessInfo("ProcessInitiator",F25,178)
```

In this example, the PlanCode parameter is used to return the process initiator for plan file 178. This approach is only supported when the process is a plan file process definition. Also, a cell reference is used for the process definition ID.

GetProcessInfo("ProcessName")

If "ProcessName" is specified as the first parameter, GetProcessInfo returns the name of a process.

▶ Syntax

```
GetProcessInfo("ProcessName", ProcessDefinitionID)
```

Parameter	Description
ProcessName	Use the keyword `ProcessName` to return the name of a process.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If you are reporting against a historical process instance, the process name is appended with an internal version number for that instance. For example, if the process name is "Rollover" then the historical process name might be something like "Rollover3."
- GetProcessInfo is a volatile function.

▶ Examples

```
=GetProcessInfo("ProcessName",5589)
```

This example returns the name of the designated process. For example: Annual Rollover.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

GetProcessInfo("StepCompletedDate")

If "StepCompletedDate" is specified as the first parameter, GetProcessInfo returns the completed date of a step in a process, given the step number and (if necessary) the plan code.

```
GetProcessInfo("StepCompletedDate", ProcessDefinitionID, PlanCode,
"StepNumber")
```

Parameter	Description
StepCompletedDate	Use the keyword `StepCompletedDate` to return the completed date of a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2.

NOTE: Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10. |

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If the process is not currently active, GetProcessInfo returns the text "Not Started." If the process is active but the step is not yet completed, GetProcessInfo returns the text "Not Completed" (this is also returned when querying historical instances if a step was never completed).
- If a step was skipped, the step completed date is the date the step was skipped.
- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("StepCompletedDate",5589,,"5")`

This example returns the completed date of the designated step. For example: 9/25/2018 if that is the date step 5 was completed. Note that the 3rd parameter must be included as a blank value if it does not apply.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

```
=GetProcessInfo("StepCompletedDate",F25,42000,"5")
```

In this example, the PlanCode parameter is used to return the completed date of step 5 for plan file 42000. Also, a cell reference is used for the process definition ID.

GetProcessInfo("StepDueDate")

If "StepDueDate" is specified as the first parameter, GetProcessInfo returns the due date of a step, given the step number and (if necessary) the plan code.

▶ Syntax

```
GetProcessInfo("StepDueDate", ProcessDefinitionID, DimensionValue,
"StepNumber")
```

Parameter	Description
StepDueDate	Use the keyword `StepDueDate` to return the due date of a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
DimensionValue	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses). If the dimension value is relevant to the step, it can be included or omitted as follows: • If no value is specified, the function returns the due date as defined for the step. This could be a fixed due date ("5/20/2018") or a relative due date ("5 days after step becomes active"). • If a value is specified, the function returns the specific due date for that value. This only matters if the due date is relative (if the due date is fixed, then it is the same for all dimension values). The step must have been started for the specified value in order to return the value-specific due date; otherwise the relative due date text is returned. For example, the function will return "10/5/2018" if the step has been started for the specified value, and will return "5 days after step becomes active" if it has not.
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- The due date is taken from the currently active process, or if the process is not active, from the process definition. If you are reporting against a historical process instance, the due date is taken from that archived instance.
- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("StepDueDate",5589,,"5")`

This example returns the due date of the designated step. For example: 9/25/2018 if that is the date that step 5 is due. Note that the 3rd parameter must be included as a blank value if it does not apply.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("StepDueDate",F25,42000,"5")`

In this example, the DimensionValue parameter is used to return the date step 5 is due for plan file 42000. Also, a cell reference is used for the process definition ID.

GetProcessInfo("StepName")

If "StepName" is specified as the first parameter, GetProcessInfo returns the name of a step in a process, given the step number.

▶ Syntax

`GetProcessInfo("StepName", ProcessDefinitionID, PlanCode, "StepNumber")`

Parameter	Description
StepName	Use the keyword `StepName` to return the step name of a step in a process.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	This parameter does not apply when using the StepName keyword and must be left blank.

Parameter	Description
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- The step name is taken from the currently active process, or if the process is not active, from the process definition. If you are reporting against a historical process instance, then the step name is taken from that archived instance.
- GetProcessInfo is a non-volatile function.

▶ Examples

```
=GetProcessInfo("StepName",5589,,"5")
```

This example returns the name of the designated step. For example: "Update current periods" if that is the name of step 5. Note that the inapplicable 3rd parameter must be included as a blank value.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

GetProcessInfo("StepOwner")

If "StepOwner" is specified as the first parameter, GetProcessInfo returns the assigned owner of a step in a process, given the step number and (if necessary) the plan code.

▶ Syntax

```
GetProcessInfo("StepOwner", ProcessDefinitionID, PlanCode, "StepNumber",
ResolveAssignments, ResolveRolesToUsers)
```

Parameter	Description
StepOwner	Use the keyword `StepOwner` to return the owner of a step.

Parameter	Description
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep.SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.
ResolveAssignments	Optional. Specifies whether the owner assignment or the resolved owner is returned. • If FALSE (default), the owner assignment is returned. This is the assignment as it is set in the process definition—meaning, a user or role name, an assignment column name, a workbook name, or the text **Process Initiator**. • If TRUE, then any dynamic owner assignments are resolved to a user name or a role name. This applies if the owner assignment is anything other than a user or role name. For example, assignment columns are resolved to the entry in the column, and workbook assignments are resolved to the entry in the workbook. This option only applies to future steps that have not yet been made active. If a step has been made active, then the actual owner of that step (user or role) is always returned.
ResolveRolesToUsers	If the assignment is a role, specifies whether the function will return the role name or a list of the individual owners within the role. • If FALSE (default), the role name is returned. • If TRUE, then the individual owners within the role are returned as a comma-separated list. This can potentially be a large list.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

► Remarks

- The step owner is taken from the currently active process, or if the process is not active, from the process definition. If you are reporting against a historical process instance, then the step owner is taken from that archived instance.

- When using GetProcessInfo to return step owners for future steps, the owners are resolved once per file session, when the function is first calculated. If the assignment changes during the current file session, the function result will not update for this change.

- If the designated step number is a type that does not take an assigned owner (for example, a Parallel Subprocess step), then the function returns "N/A".

- If the designated step was skipped and therefore there was no step owner, the function returns "<skipped>".

- GetProcessInfo is a non-volatile function.

► Examples

```
=GetProcessInfo("StepOwner",5589,,"5")
```

This example returns the assigned owner of the designated step. For example: "Jane Doe" if that user is the assigned owner of step 5. Note that the 3rd parameter must be included as a blank value if it does not apply.

This assumes that the assigned owner is a specified user. If instead the step uses a dynamic assignment option, such as an assignment column, then this function would return the configured assignment (such as "Dept.Owner" or "Assignment.xlsx").

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

```
=GetProcessInfo("StepOwner",F25,42000,"5",True)
```

In this example, the PlanCode parameter is used to return the assigned owner of step 5 for plan file 42000. The ResolveAssignments parameter is also used to resolve any dynamic ownership assignments. For example, if the configured owner assignment is a column, the function will return the owner in that column such as "Jane Doe".

If the resolved owner name in the column is a role such as "Finance," then that role name is returned, not the individual owners in the role. If you want the function to return the individual owners in the role, then you would use the ResolveRolesToUsers parameter like so: `=GetProcessInfo ("StepOwner",F25,42000,"5",True,True)` . This applies whenever a role name is returned by the function, regardless of whether the role name is the configured assignment or the resolved owner name.

This example also uses a cell reference for the process definition ID.

GetProcessInfo("StepType")

If "StepType" is specified as the first parameter, GetProcessInfo returns the step type of a step, given the step number.

▶ Syntax

`GetProcessInfo("StepType", ProcessDefinitionID, PlanCode, "StepNumber")`

Parameter	Description
StepType	Use the keyword `StepType` to return the step type of a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	This parameter does not apply when using the StepType keyword and must be left blank.
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- The step type is taken from the currently active process, or if the process is not active, from the process definition. If you are reporting against a historical process instance, then the step type is taken from that archived instance.
- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("StepType",5589,,"5")`

This example returns the type of the designated step. For example: "Generic Process Step" if that is the type of step 5. Note that the inapplicable 3rd parameter must be included as a blank value.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

GetProcessInfo("StepStartDate")

If "StepStartDate" is specified as the first parameter, GetProcessInfo returns the start date of a step in a process, given the step number and (if necessary) the plan code.

▶ Syntax

`GetProcessInfo("StepStartDate", ProcessDefinitionID, PlanCode, "StepNumber")`

Parameter	Description
StepStartDate	Use the keyword `StepStartDate` to return the start date of a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If the step is not started, GetProcessInfo returns the text "Not Started." This applies whether the process is currently active or not, and also if you are reporting against a historical instance where the step was never started.
- If the step was skipped, the step start date is the date the step was skipped.
- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("StepStartDate",5589,,"5")`

This example returns the start date of the designated step. For example: "9/19/2018" if that is the date and time step 5 was started. Note that the 3rd parameter must be included as a blank value if it does not apply.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

```
=GetProcessInfo("StepStartDate",F25,42000,"5")
```

In this example, the PlanCode parameter is used to return the start date of step 5 for plan file 42000. Also, a cell reference is used for the process definition ID.

GetProcessInfo("StepStatus")

If "StepStatus" is specified as the first parameter, GetProcessInfo returns the status of a step in a process, given the step number and (if necessary) the plan code.

▶ Syntax

```
GetProcessInfo("StepStatus", ProcessDefinitionID, PlanCode, "StepNumber")
```

Parameter	Description
StepStatus	Use the keyword `StepStatus` to return the status of a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2.

NOTE: Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10. |

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- Return values are: Active, Completed, Skipped, Stalled, Aborted, and Not Started. Note that if a step is bypassed by use of the administration feature Move Current Step, then that step shows as Not Started (instead of Completed or Skipped).
- GetProcessInfo is a non-volatile function.

`=GetProcessInfo("StepStatus",5589,,"5")`

This example returns the status of the designated step. For example: "Active" if step 5 is currently active. Note that the 3rd parameter must be included as a blank value if it does not apply.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("StepStatus",F25,42000,"5")`

In this example, the PlanCode parameter is used to return the status of step 5 for plan file 42000. Also, a cell reference is used for the process definition ID.

GetProcessInfo("StepStatusDetails")

If "StepStatusDetails" is specified as the first parameter, GetProcessInfo returns the status details of a step in a process, given the step number and (if necessary) the plan code. The step details returned by the function are similar to the details shown in the Process Status dialog or the Process Routing page.

▶ Syntax

`GetProcessInfo("StepStatusDetails", ProcessDefinitionID, PlanCode, "StepNumber", ResolveAssignments, ResolveRolesToUsers)`

Parameter	Description
StepStatusDetails	Use the keyword `StepStatusDetails` to return the step details.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep.SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2.

NOTE: Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10. |

Parameter	Description
ResolveAssignments	Optional. If the details return future step ownership, specifies whether the owner assignment or the resolved owner is returned.
	• If FALSE (default), the owner assignment is returned. This is the assignment as it is set in the process definition—meaning, a user or role name, an assignment column name, a workbook name, or the text **Process Initiator**.
	• If TRUE, then any dynamic owner assignments are resolved to a user name or a role name. This applies if the owner assignment is anything other than a user or role name. For example, assignment columns are resolved to the entry in the column, and workbook assignments are resolved to the entry in the workbook.
ResolveRolesToUsers	Optional. If the details return an owner assignment, and the assignment is a role, specifies whether the function will return the role name or a list of the individual owners within the role.
	• If FALSE (default), the role name is returned.
	• If TRUE, then the individual owners within the role are returned as a comma-separated list. This can potentially be a large list.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If the specified step is a past step, the details indicate whether the step was skipped, completed, or never active.

- If the specified step is the current step, the details vary as follows:
  - For regular steps, the details indicate the currently assigned users for the step. If the step is stalled, the details indicate the error.
  - For subprocess steps, the details indicate the number of substeps that have been completed in the subprocess. If one of the subprocess steps is stalled, the details indicate the stalled status.

- If the specified step is a future step, the details indicate the owner assignment for the step as a future assignment. The function behavior and the return value is like when using the StepOwner keyword. You can choose whether to resolve dynamic assignments to users or roles, and whether to resolve role assignments to individual users in the role.

- If the process is not currently active, then no step details are returned, unless you are using a historical process ID to return information on a previous process execution.

- GetProcessInfo is a non-volatile function.

`=GetProcessInfo("StepStatusDetails",5589,,"5")`

This example returns the detail text for the specified step. The returned details might vary as follows:

- If step 5 is a past step, the detail text would be something like "Completed on 6/16/2017 by Jane Doe".
- If step 5 is the current step, the detail text would be something like "Assigned to user Jane Doe".
- If step 5 is a future step, the detail text would be something like "Future assignment: Jane Doe".

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("StepStatusDetails",F25,42000,"5")`

In this example, the process is a plan file process, so the PlanCode parameter is used to return the detail text for plan file 42000, for step 5. The returned details might vary as follows:

- If step 5 is a past step, the detail text would be something like "Completed on 6/16/2017 by Jane Doe" or "Skipped on 6/16/2017".
- If step 5 is the current step, the detail text would be something like "Assigned to user Jane Doe".
- If step 5 is a future step, the detail text would be something like "Future assignment: Dept.Owner". If the assignment is a dynamic assignment, the Resolve Assignments parameter could be used to resolve the assignment to a specific user or role instead, and the ResolveRolesToUsers parameter could be used to resolve a role name to specific users in the role.

This example also uses a cell reference for the process definition ID.

GetProcessInfo("StepCompletedBy")

If "StepCompletedBy" is specified as the first parameter, GetProcessInfo returns the user who completed a step in a process, given the step number and (if necessary) the plan code.

▶ Syntax

`GetProcessInfo("StepCompletedBy", ProcessDefinitionID, PlanCode, "StepNumber")`

Parameter	Description
StepCompletedBy	Use the keyword `StepCompletedBy` to return the user who completed a step.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).

Parameter	Description
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep*.*SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2.
	NOTE: Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If the process is not currently active, GetProcessInfo returns the text "Not Started." If the process is active but the step is not yet completed, GetProcessInfo returns the text "Not Completed" (this is also returned when querying historical instances if a step was never completed).
- Returns "<skipped>" if the step was skipped.
- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("StepCompletedBy",5589,,"5")`

This example returns the user who completed the designated step. For example: "Jane Doe" if that is the user who completed step 5. Note that the 3rd parameter must be included as a blank value if it does not apply.

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("StepCompletedBy",F25,42000,"5")`

In this example, the PlanCode parameter is used to return the user who completed step 5 for plan file 42000. Also, a cell reference is used for the process definition ID.

GetProcessInfo("TimeinStep")

If "TimeinStep" or "TimeinStepRaw" is specified as the first parameter, GetProcessInfo returns the time spent in the step, given the step number and (if necessary) the plan code.

▶ Syntax

`GetProcessInfo("TimeinStep", ProcessDefinitionID, PlanCode, "StepNumber")`

Parameter	Description
TimeInStep	Use the keyword `TimeinStep` to return the time spent in the step. By default, this is returned as a text value, such as "2 days". Alternatively, you can use the keyword `TimeinStepRaw` to return the raw time value, in seconds.
ProcessDefinitionID	The database ID of the process definition. You can find this value for a process by using GetProcessInfo("ProcessDefinitionID").
PlanCode	The plan code for which you want to return information. Only applies to plan file processes (both plan file process definitions and plan file subprocesses).
StepNumber	The number of the step in the process. If the step is a sub-step, enter the step number as *ParentStep.SubStep*. For example: enter 2.5 if the step is the fifth sub-step under step 2. **NOTE:** Although this parameter takes a number, it is recommended to place the number in quotation marks so that it will be read as a string. For example, if you enter 2.10 as a number, Excel will drop the trailing 0 and use 2.1 instead. If instead the value is entered as "2.10" then it will always be read as 2.10.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- If the step has not yet been active, TimeinStep returns blank, and TimeinStepRaw returns zero. If the step is active but for less than 1 hour, TimeinStep returns "0 Hours".

- If the process is not currently active, then no time in step details are returned, unless you are using a historical process ID to return information on a previous process execution.

- GetProcessInfo is a non-volatile function.

▶ Examples

`=GetProcessInfo("TimeinStep",5589,,"5")`

This example returns the time the process spent in the designated step. For example, "2 Days".

Instead of "hard-coding" the process definition ID within the function, in most cases you will return the ID once in a reserved cell and then reference that cell in all GetProcessInfo functions where you want to return information about that process.

`=GetProcessInfo("TimeinStep",F25,42000,"5")`

This example is for a plan file process, so the PlanCode parameter is used to return the time that plan file 42000 spent in step 5. For example, "17 Hours". Also, a cell reference is used for the process definition ID.

```
=GetProcessInfo("TimeinStepRaw",F25,42000,"5")
```

This example is the same as the previous example, except that the keyword TimeinStepRaw is used to return the raw time value in seconds. For example: "62000". You can then use formulas in the spreadsheet to convert this value as desired.

GetSecurityInfo function

Returns information on a user's security settings. The function has different syntax depending on the keyword used in the first parameter, which determines the type of security information to be returned.

> **NOTE:** You can use a data lookup to return the same information as a GetSecurityInfo function. Whenever possible, it is recommended to use data lookup queries over Axiom functions for faster performance.

The user's inherited role permissions and the user's administrator status are applied to the GetSecurityInfo results. For example, if the user has a defined table type filter, but the user is also an administrator, then the GetSecurityInfo("TableType") function will return full access, because filters are not applied to administrators. Subsystem restrictions are also applied, if applicable.

You can use the GetSecurityInfo function to return the following information:

Item	See
File group access	GetSecurityInfo("FileGroup")
Table type access	GetSecurityInfo("TableType")
Table access	GetSecurityInfo("Table")
System administrator rights	GetSecurityInfo("IsSysAdmin")
Security permissions	GetSecurityInfo("PermissionGranted")
Role assignments	GetSecurityInfo("InRole")

GetSecurityInfo("FileGroup")

If "FileGroup" is specified as the first parameter, GetSecurityInfo returns the user's plan file access filter, given a file group.

▶ Syntax

```
GetSecurityInfo("FileGroup", "FileGroupName", "UserName", LocalizeResult,
"Domain")
```

Parameter	Description
FileGroup	Use the keyword `FileGroup` to return information on a user's file group access filter.
FileGroupName	The name of the file group. A file group alias name can also be used.
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. Boolean value to determine whether the plan file 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.
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.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- The second parameter of this function also accepts a file group code. However, use of file group code is being phased out. If you have existing functions that use the file group code, these functions should eventually be migrated to use the file group name.

- This function 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 process or workflow elevation.

- 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.

- If the filter uses a filter variable (such as `{CurrentUser.LoginName}`), the variable is resolved when the function returns values for a user, and not resolved when the function returns values for a role.

- GetSecurityInfo is a volatile function.

▶ Examples

`=GetSecurityInfo("FileGroup","Budget 2019")`

This example returns "1" if the current user has full access to file group Budget 2019.

`=GetSecurityInfo("FileGroup","Budget 2019",,TRUE)`

This example returns "Full Access" if the current user has full access to file group Budget 2019.

`=GetSecurityInfo("FileGroup","Budget 2019","Jdoe")`

This example returns "(DEPT.Region='North')" if user Jdoe has filtered access to file group Budget 2019.

`=GetSecurityInfo("FileGroup","Budget 2019","Jdoe",,"Corporate")`

This example returns the file group filter for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

GetSecurityInfo("TableType")

If "TableType" is specified as the first parameter, GetSecurityInfo returns the user's table type permissions, given a table type.

▶ Syntax

`GetSecurityInfo("TableType", "TableTypeName", "UserName", LocalizeResult, "Domain", ReturnWriteFilter)`

Parameter	Description
TableType	Use the keyword `TableType` to return information on the user's table type access filter.
TableTypeName	The name of the table type.
UserName	Optional. The name of the user or role for which to return the security information. If omitted, the current user is assumed.

Parameter	Description
LocalizeResult	Optional. Boolean value to determine whether the table type 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.
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.
ReturnWriteFilter	Optional. Boolean value to determine whether the function returns the user's read permissions or the user's write permissions. • If FALSE (default), the function returns the user's read permissions to the table type. • If TRUE, the function returns the user's write permissions to the table type. 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 to the table type. However, if you need to return the write access specifically, you can use this parameter to do so.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE should not be in quotation marks.

▶ Remarks

- The user's effective permissions are returned, including any access inheritance from a role or admin rights, and subsystem restrictions (if applicable).

- If the filter uses a filter variable (such as `{CurrentUser.LoginName}`), the variable is resolved when the function returns values for a user, and not resolved when the function returns values for a role.

- GetSecurityInfo is a volatile function.

Examples

`=GetSecurityInfo("TableType","GLData")`

This example returns "0" if the current user has no read access to table type GLData.

`=GetSecurityInfo("TableType","GLData",,TRUE)`

This example returns "No Access" if the current user has no read access to table type GLData.

`=GetSecurityInfo("TableType","GLData","Jdoe")`

This example returns "(DEPT.Region='North')" if user Jdoe has filtered read access to table type GLData.

`=GetSecurityInfo("TableType","GLData","Jdoe",,"Corporate")`

This example returns the table type filter for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

`=GetSecurityInfo("TableType","GLData",,TRUE,,TRUE)`

This example returns "No Access" if the current user has no write access to table type GLData. The user might also have no read access to the table type, or the user might have full or filtered read access to the table type but no write access to the table type. The sixth parameter is used to specifically return the write access, regardless of what the user's read access is.

GetSecurityInfo("Table")

If "Table" is specified as the first parameter, GetSecurityInfo returns the user's table access permissions, given a table name.

▶ Syntax

`GetSecurityInfo("Table", "TableName", "UserName", LocalizeResult, "Domain", ReturnWriteFilter)`

Parameter	Description
Table	Use the keyword `Table` to return information on the user's table access permissions.
TableName	The name of the table.
UserName	Optional. The name of the user or role for which to return the security information. If omitted, the current user is assumed.

Parameter	Description
LocalizeResult	Optional. Boolean value to determine whether the table 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.
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.
ReturnWriteFilter	Optional. Boolean value to determine whether the function returns the user's read permissions or the user's write permissions. • If FALSE (default), the function returns the user's read permissions to the table. • If TRUE, the function returns the user's write permissions to the table. 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 to the table. However, if you need to return the write access specifically, you can use this parameter to do so.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE should not be in quotation marks.

▶ Remarks

- The user's effective permissions are returned, including any access inheritance from a table type, role, or admin rights, and subsystem restrictions (if applicable).

- If the filter uses a filter variable (such as `{CurrentUser.LoginName}`), the variable is resolved when the function returns values for a user, and not resolved when the function returns values for a role.

- GetSecurityInfo is a volatile function.

`=GetSecurityInfo("Table","GL2018")`

This example returns "0" if the current user has no read access to table GL2018.

`=GetSecurityInfo("Table","GL2018",,TRUE)`

This example returns "No Access" if the current user has no read access to table GL2018.

`=GetSecurityInfo("Table","GL2018","Jdoe")`

This example returns "(DEPT.Region='North')" if user Jdoe has filtered read access to table GL2018.

`=GetSecurityInfo("Table","GL2018","Jdoe",,"Corporate")`

This example returns the table access permissions for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

`=GetSecurityInfo("Table","Dept",,TRUE,,TRUE)`

This example returns "No Access" if the current user has no write access to table Dept. The user might also have no read access to the table, or the user might have full or filtered read access to the table but no write access to the table. The sixth parameter is used to specifically return the write access, regardless of what the user's read access is.

GetSecurityInfo("IsSysAdmin")

If "IsSysAdmin" is specified as the first parameter, GetSecurityInfo returns whether the user has administrator permissions to the current system (TRUE/FALSE).

A user is an administrator of the current system if they have the **Administrator** permission.

▶ Syntax

`GetSecurityInfo("IsSysAdmin", "UserName", "Domain")`

Parameter	Description
IsSysAdmin	Use the keyword `IsSysAdmin` to return the user's system administrator status.
UserName	Optional. The name of the user. If omitted, the current user is assumed.
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.

Remarks

GetSecurityInfo is a volatile function.

Examples

`=GetSecurityInfo("IsSysAdmin")`

This example returns "TRUE" if the current user is a system administrator.

`=GetSecurityInfo("IsSysAdmin","Jdoe")`

This example returns "FALSE" if user Jdoe is not a system administrator.

`=GetSecurityInfo("IsSysAdmin","Jdoe","Corporate")`

This example returns the system administrator status for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

GetSecurityInfo("PermissionGranted")

If "PermissionGranted" is specified as the first parameter, GetSecurityInfo returns whether the user has a specific security permission (TRUE/FALSE), given a permission name. Security permissions are defined on the **Permissions** tab of the **Security** dialog.

Syntax

`GetSecurityInfo("PermissionGranted", "PermissionName", "UserName","Domain")`

Parameter	Description
PermissionGranted	Use the code `PermissionGranted` to return the user's security permission status.
PermissionName	The name of the permission to check. See the following table for specific values.
UserName	Optional. The name of the user or role for which to return the security information. If omitted, the current user is assumed.
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.

The parameter PermissionName takes the following values:

Security Permission	Value To Use In GetSecurityInfo
Administer Announcements	AdministerAnnouncements
Administer Axiom Explorer	AdministerRepository
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

GetSecurityInfo is a volatile function.

▶ Examples

`=GetSecurityInfo("PermissionGranted","ScheduledJobsUser")`

This example returns "TRUE" if the current user has the **Scheduled Jobs User** permission.

`=GetSecurityInfo("PermissionGranted","ScheduledJobsUser","Jdoe")`

This example returns "FALSE" if user Jdoe does not have the **Scheduled Jobs User** permission.

`=GetSecurityInfo("PermissionGranted","ScheduledJobsUser","Jdoe","Corporate")`

This example returns information for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

GetSecurityInfo("InRole")

If "InRole" is specified as the first parameter, GetSecurityInfo returns whether the user belongs to the specified role (TRUE/FALSE).

Syntax

```
GetSecurityInfo("InRole", "Role", "UserName", "Domain")
```

Parameter	Description
InRole	Use the keyword `InRole` to return the user's role assignment status.
Role	The name of the role to check.
UserName	Optional. The name of the user. If omitted, the current user is assumed.
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.

Remarks

GetSecurityInfo is a volatile function.

Examples

```
=GetSecurityInfo("InRole","Finance")
```

This example returns "TRUE" if the current user belongs to the Finance role.

```
=GetSecurityInfo("InRole","Finance","Jdoe")
```

This example returns "FALSE" if user Jdoe does not belong to the Finance role.

```
=GetSecurityInfo("InRole","Finance","Jdoe","Corporate")
```

This example returns the role status for the user Jdoe who was imported from the Corporate domain. This would be necessary if another user named Jdoe was imported from a different Active Directory domain.

GetSystemInfo function

Returns information about the current Axiom Software system.

Syntax

```
GetSystemInfo("SystemInfoCode")
```

The following codes can be used for the parameter SystemInfoCode:

Code	Description
AppserverURI	Returns the application server URI for the current Axiom Software installation.
ClientType	Returns the current client context for the Axiom file, such as Excel, Windows, Web, iOS, Scheduler, and Server. **NOTES:** • If you view an Axiom form in the Excel Client or Windows Client, the result of this function is "Web" because the file is being viewed as a web page. In this circumstance, the Web Client is effectively operating within the Excel or Windows Client. • If you view an Axiom form on an iPad, the result of this function is "iOS" if the form was opened using the Axiom iPad app. If the form was opened using the Safari browser on the iPad, then the result of this function is "Web". • "Server" is returned when a workbook is being processed in order to service a system request made to the application server. Currently this only applies to process management when reading assignments from assignment workbooks.
CurrentLanguage	Returns the two-letter code for the language of the current client session.
DBMS	Returns the type of database being used with the system (for example: SQLServer).
HelpURL	Opens an Axiom Software Help topic, given a context-sensitive help ID. For more information, see Using GetSystemInfo to display help topics.
EngineType	Returns the spreadsheet engine being used for processing the Axiom file: Excel or Web.
ProductVersion	Returns the Axiom Software version number.
SystemName	Returns the name of the system that the user is currently logged into.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetSystemInfo is a non-volatile function.

▶ Examples

`=GetSystemInfo("SystemName")`

This example returns the name of the current Axiom Software system. For example: "Portland Bike Strategic Planning".

`=GetSystemInfo("ProductVersion")`

This example returns the Axiom Software version number. For example: "2018.1.10.21".

`=GetSystemInfo("ClientType")`

This example returns the current Axiom Software Client type. For example: "Web".

`=GetSystemInfo("AppserverURI")`

This example returns the URI to the application server for the current Axiom Software installation. For example: "http://ServerName/Axiom".

`=GetSystemInfo("CurrentLanguage")`

This example returns the code for the language of the current client session. For example: "en" if it is English.

▶ Using GetSystemInfo to display help topics

You can use GetSystemInfo within a Hyperlink function to link to specific topics in the Axiom Software help files. You may want to link to a topic from within a file to provide the file's users with easy access to more information about a particular subject. You can link to the general Axiom Software help files, or to product-specific help files.

To do this, use the following syntax for GetSystemInfo:

`GetSystemInfo("HelpURL","TopicID","HelpType","Product")`

Parameter	Description
HelpURL	Use the keyword `HelpURL` to generate a URL to the help files.
TopicID	Optional. The ID of a specific help topic. If a topic ID is specified, the URL opens that help topic.
	For the general (platform) Axiom Software help file, the topic ID is the file name of the topic, minus the file extension. For example, if the file name for the topic is "Batch.htm", then you would enter "Batch" as the topic ID. The file name for each topic can be found in the lower right-hand corner of the topic, in the topic footer. Product help files may use different IDs.
	If the topic ID is omitted, or if the specified ID cannot be found in the target help system, the URL opens the default home page for the target help system.

Parameter	Description
HelpType	Optional. The type of help system to open. You can specify the following: • `Desktop`: Opens the Desktop Client help file for the Axiom Software platform. This is the default option if the parameter is omitted. • `Web`: Opens the Web Client help file for the Axiom Software platform. • `Product`: Opens a product help file, as specified in the Product parameter.
Product	The name of an Axiom Software product, such as Capital Planning or Budget Planning. This is required if `Product` is specified in the HelpType parameter, and ignored otherwise. The product name must exactly match the installed feature name.

The GetSystemInfo function generates a URL, which can then be placed within Excel's Hyperlink function to generate a clickable hyperlink in a file. For example:

```
=HYPERLINK(GetSystemInfo("helpurl","Batch"),"Get help on batch
processing")
```

The generated URL automatically points to the help file for the current platform version or product version.

> **NOTES:**
>
> • Although Axiom will make every effort to maintain existing help IDs, there is no guarantee that help IDs will remain constant from release to release. The introduction of new features and enhancements in a release may necessitate reorganizing existing information, which may include changing help IDs or moving information to different topics.
>
> • This function is intended to be used within an Axiom file, to dynamically generate a live URL that a user can consume within the file. It is not intended to generate a persistent URL that can be copied and pasted to an external location and used at any time.

GetTableInfo function

Returns information about a table, given the table name.

▶ Syntax

```
GetTableInfo("CodeName", "TableName")
```

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. • `IndexScheme`: Returns the index scheme for the table, such as Default or Large Table. This feature is only available to systems running Axiom Software version 2018.01.54 or higher. • `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. • `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.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetTableInfo is a non-volatile function.

▶ Examples

`=GetTableInfo("Classification","Dept")`

This example returns the table classification of Dept. For example: Reference.

`=GetTableInfo("TableType","GL2018")`

This example returns the table type that the table is assigned to. For example: GLData.

GetURLEncodedString function

Returns an encoded string that can be included in a URL.

This function is intended to be used when you are creating a URL by combining two or more strings via formula. If the strings to be included in the URL include spaces or other special characters that require encoding, this function can be used to encode the strings so that the resulting URL is valid.

```
GetURLEncodedString("String")
```

Parameter	Description
String	The string to be encoded for use in a URL.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- GetURLEncodedString is a non-volatile function.

▶ Examples

```
=GetURLEncodedString("Axiom Software")
```

This example returns the encoded string "Axiom%20Software." The encoded string is now valid to be used within a URL.

GetUserInfo function

Returns information about an Axiom Software user, as defined in Security.

> **NOTE:** You can use a data lookup to return the same information as a GetUserInfo function. Whenever possible, it is recommended to use data lookup queries over Axiom functions for faster performance.

▶ Syntax

```
GetUserInfo("CodeName", "UserName", "UserID", "Domain")
```

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 Software). • `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.
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.

- 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.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetUserInfo is a volatile function.

▶ Examples

`=GetUserInfo("UserName")`

This example returns the login name for the current user. For example: "jdoe".

`=GetUserInfo("FirstName")`

This example returns the first name of the current user. For example: "John".

```
=GetUserInfo("UserID","jsmith")
```

This example uses the optional second parameter to return the user ID of the user jsmith. For example: "17".

```
=GetUserInfo("UserName",,17)
```

This example uses the optional third parameter to return the login name of the user with ID 17. For example: "jsmith".

```
=GetUserInfo("UserEmail","jsmith",,"Corporate")
```

This example returns the email address of user jsmith who was imported from the Corporate domain. This would be necessary if another user named jsmith was imported from a different Active Directory domain.

Document Information Functions

This section contains information on functions that return information about the current document.

GetColumnInfo function

Returns information about a table column, given the column name.

▶ Syntax

`GetColumnInfo("CodeName", "ColumnName")`

Parameter	Description
CodeName	Specifies the column property to return. Use one of the following values: • `Alias`: Returns the Table.Column associated with an alias name. The ColumnName parameter must be a valid alias name. • `DataType`: Returns the data type of the column, such as String(20) or Integer. • `IsCalculated`: Returns whether the column is a calculated column (True/False). The following values only apply to product systems, and are configured by product development and implementation consultants: • `IsUnused`: Returns whether the column is flagged as unused (True/False). • `IsVariable`: Returns whether the column is flagged as variable (True/False). • `PreferredName`: Returns the preferred name of the given Table.Column.
ColumnName	The name of the column for which you want to return information. Use Table.Column syntax, such as "GL2018.M1". Multi-level column syntax can be used. When using the Alias code, the column name should be the alias name by itself, such as "CYB1".

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetColumnInfo is a non-volatile function.

▶ Examples

```
=GetColumnInfo("DataType","Dept.Dept")
```

This example returns the data type of Dept.Dept. For example: Integer.

```
=GetColumnInfo("Alias","CYA1")
```

This example returns the column name associated with the CYA1 alias. For example: GL2018.M1.

```
=GetColumnInfo("IsCalculated","GL2018.TOT")
```

In this example, the specified column is a calculated column, so the function returns True.

GetColumnLetter function

Returns the column letter of the current cell, of the active cell when the file was refreshed, or for a specific cell reference.

The intent of GetColumnLetter is to help support custom drilling in Axiom files using double-click.

▶ Syntax

```
GetColumnLetter("Cell")
```

The parameter Cell takes the following values:

- <Blank>: Leave the Cell parameter blank (open parentheses) to return the column letter for the current cell (the cell that contains the GetColumnLetter function).
- `Active`: Use this keyword to return the column letter for the cell that was active when the last recalculation occurred.
- *CellReference*: Enter a cell address to return the column letter for that cell.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- Drilling initiates a recalculation, so `Active` can be used to identify the column where the user double-clicked to perform the drill.
- GetColumnLetter is a volatile function.

`=GetColumnLetter()`

This example returns the column letter for the cell that contains the GetColumnLetter function (for example: "A").

`=GetColumnLetter("Active")`

This example returns the column letter for the active cell when the file was refreshed (for example: "F").

`=GetColumnLetter("AD25")`

This example returns the column letter for the specified cell address ("AD" in this case).

GetCurrentSheetView function

Returns the names of the current views applied to a sheet. The function can return the sheet view, column views, and row views.

▶ Syntax

`GetCurrentSheetView("ViewType", "SheetName")`

Parameter	Description
ViewType	Use one of the following keywords to specify which views you want to return: • `AppliedSheetView`: Returns the name of the currently applied sheet view. • `AppliedColumnView`: Returns the names of the currently applied column views. • `AppliedRowView`: Returns the names of the currently applied row views. All of these items return blank if no view is currently applied. For column and row views, if multiple views are applied then the names are returned as a comma-separated list.
SheetName	The name of the sheet for which you want to return the current view names. The sheet must be configured on the Control Sheet in order to track and return currently applied views.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

• This function is intended for configurations where you need to know the currently applied views for sheets that were dynamically added using the master sheets feature. However, when you are not using master sheets, then you should not use this function. Instead, you can write a formula

that references the applicable **Current Dynamic View** field on the Control Sheet (there are two additional fields for **Current Dynamic Column Views** and **Current Dynamic Row Views**). These fields are always updated to report the currently applied views for each sheet.

- GetCurrentSheetView is a volatile function.

▶ Examples

`=GetCurrentSheetView("AppliedSheetView","Proposal")`

This example returns the name of the currently applied sheet view for the Proposal sheet, such as "All Detail".

`=GetCurrentSheetView("AppliedColumnView","Report")`

This example returns the names of the currently applied column views for the Report sheet, such as "Q4" (if only one column view is applied) or "Q3, Q4" (if multiple column views are applied).

GetDocumentHyperlink function

Creates a hyperlink to a file in the Axiom Software file system, which can be used to:

- Launch the Axiom Software system
- Automatically open a specified file

The intent of this function is to create a hyperlink that is then emailed or otherwise given to a user so that they can open the file directly, without needing to manually open Axiom Software and find the file. If you simply want to link to an Axiom file from within another Axiom file, use the GetDocument function instead.

GetDocumentHyperlink can also be used to create a URL for use in Axiom forms (for example, in content tags for Formatted Grid components).

> **NOTE:** Users of the Axiom shared client (on Citrix or Terminal Server) cannot use document hyperlinks to open Axiom files.

> **TIP:** Alternatively, you can right-click a document in Axiom Explorer to copy a URL for that file to your clipboard.

▶ Syntax

`GetDocumentHyperlink("DocumentPathorID", "ClientType", "SheetFilter", "Language", "CellAddress")`

Parameter	Description
DocumentPathorID	Optional. The file for which you want to create a hyperlink. If omitted, the current file is assumed.
	You can specify a file other than the current file by entering either of the following:
	• The document ID of the file, such as 93.
	• The full path to the file, such as `\Axiom\Reports Library\Monthly Reports\Income Statement.xlsx`.
ClientType	Optional. The client context in which to open the file. For more information, see the following section on client type options.
SheetFilter	Optional. A filter to be applied as a temporary sheet filter to the target file.
	This parameter specifies both the type of filter (table type, table, etc.) and the filter criteria statement. See the following section for more information on the sheet filter syntax.
	When the target file is opened via hyperlink, the specified filter is applied as a sheet filter behind the scenes. The filter does not display on the Control Sheet, and will not be saved in the file (similar to Quick Filter functionality). If desired, the user can clear the filter using the Quick Filter dialog.
	Keep in mind that the target file must be refreshed in order for the temporary filter to be applied to the data. One or both of the following settings should be enabled in the file:
	• **Refresh all Axiom functions on open** (if the file uses functions to return data instead of an Axiom query)
	• **Refresh data on file open** (for the applicable Axiom queries)

Parameter	Description
Language	Optional. Specifies the language to use in the Axiom Software session when the file is opened (assuming the client is not already open on the user's machine). By default, the client will open in the regional format specified by the user's operating system, if supported. This parameter can be used to override that behavior and open in the specified language—for example, to always open the file within an English language session. To specify a language, use the IETF standard language tags. The languages supported by Axiom Software use the following tags: • English: `en-US` • Dutch: `nl-NL` • French: `fr-FR` • Swedish: `sv-SE`
CellAddress	Optional. Specifies the cell to make active when the file is opened, using normal cell reference syntax. For example: `Sheet1!C3` If the specified location would not be in view normally then the file will be scrolled to that location; otherwise the file will open in its default view with the cursor placed at that location.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Client type options

If the target file for the function is a standard spreadsheet Axiom file, then you can optionally use the ClientType parameter to explicitly open the file in the Excel Client or the Windows Client using the following codes:

• `Windows`: Opens the file in the Windows Client.

• `Excel` (or `XL`): Opens the file in the Excel Client.

If it does not matter which client is used to open the file, then you can omit the parameter. In this case, the file will open in whichever client the user launched last.

If the client is specified, and the user already has that client open, then the file is simply opened within that client instance. If the user has a client open, but it does not match the specified client type, then a second Axiom Software instance is launched, using the specified client.

Other codes can be used in the ClientType parameter to support opening form-enabled Axiom files, and other special file types. When using one of these options, the Desktop Client is not launched:

- `Forms` (or `Form` or `Web`): Opens the file in the Web Client. The file must be form-enabled.

- `Resource`: Downloads the file as a resource. The file must be a plan file attachment, or other non-Axiom file (such as a Word file saved to the Reports Library). In this case the user is prompted to open the file in its native application, or save it locally.

- `Data Diagram`: Opens a Data Diagram file in the Web Client.

- `Data Explorer`: Opens a Data Explorer file in the Web Client.

If you are opening one of these special file types, then it is recommended to use the appropriate ClientType code so that the Desktop Client does not launch unnecessarily.

▶ Sheet filter syntax

The sheet filter parameter uses the following syntax:

```
FilterTarget;CriteriaStatement
```

The *FilterTarget* is a table name or a table type name. It specifies the tables to be affected by the sheet filter.

- Reference table: If the filter target is a reference table (such as DEPT), then the filter is applied to all tables that link to the DEPT table.
- Table type: If the filter target is a table type (such as GL), then the filter is applied to all tables in the table type.
- Data table: If the filter target is a data table (such as GL2018), and the table belongs to a table type, then the filter is applied to all tables in the table type. Otherwise, the filter is applied to the specified table.

> **NOTE:** Axiom Software checks for matches in the order listed above. Therefore if a reference table and a table type share the same name, the target will be the reference table.

The *CriteriaStatement* is the filter criteria statement to apply to the affected tables. Standard filter criteria syntax applied. For example:

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

This filter applies to all tables that link to the DEPT table, and filters the data for the North region.

> **NOTES:**
> - When using the sheet filter parameter, keep in mind that the temporary filter will be concatenated with any existing sheet filters in the file using AND. If the document has existing sheet filters, you should test the GetDocumentHyperlink filter to ensure that the combination of filters returns data as you intended.
> - The function GetCurrentValue can be used in the target file to return information about the temporary sheet filter.
> - If the target file is already open when a user clicks the hyperlink with a sheet filter, the file is reopened without prompting the user to close and save.

▶ Remarks

- If a new client instance is launched as a result of the hyperlink, normal startup routines apply. After the system opens, the specified document is automatically opened, in addition to the normal startup file(s) for the user.

- The user's security settings are honored to determine if the user has the right to open the specified file. Note that if the user does not have access to the file, then the Axiom Software client still launches but the file does not open. In this case, no error message is displayed to the user to let them know why the file did not open.

- If your system is deployed in an environment where users work in different system languages, then you should use system folder names in any file paths.

- The format of the generated hyperlink is different if your system uses SAML or OpenID authentication. By default, the hyperlink uses the C1 (ClickOnce) URL in order to launch the client. If you are using SAML or OpenID authentication, then the hyperlink uses the Axiom Home URL to allow for web-based authentication. This difference is not important to users except in the following cases:
  - If the authentication method of your system changes, then any hyperlinks generated using the prior authentication method may not work with the new authentication method.
  - If you are using SAML or OpenID authentication, Axiom Prompt users cannot use the hyperlinks generated in your system.

- A GetDocumentHyperlink wizard is available on the right-click context menu, and on the **Axiom Designer** tab.

- GetDocumentHyperlink is a non-volatile function.

▶ Examples

```
=GetDocumentHyperlink(311)
```

This example creates a hyperlink to launch the file that has document ID 311, using whichever desktop client was last launched on the user's machine. The following examples show how the resulting hyperlink may look:

Default hyperlink:

```
http://servername/Axiom/c1/Axiom.UI.Start.application?
docref=%3a311%3a%3a%3a%3a
```

SAML or OpenID hyperlink:

```
 http://servername/Axiom/home/launch?docref=%3a311%3a%3a%3a%3a
```

Note that both of the URLs shown above are representative of an on-premise system. If the system was a Cloud Service system, the site name would be different (like `https://CustomerName.axiom.cloud`) but the remainder of the URL would vary in the same way depending on the authentication type.

```
=GetDocumentHyperlink("\Axiom\Reports Library\Monthly Reports\Income
Statement.xlsx","Excel")
```

This example creates a hyperlink to launch the specified file using the Excel Client. In this example, the file path is used instead of the document ID.

```
=GetDocumentHyperlink(651,"Forms")
```

This example creates a hyperlink to launch the specified file as an Axiom form in the Web Client. This is the only way to open a form-enabled file as an Axiom form using the GetDocumentHyperlink function. If the ClientType parameter were left blank, or if Excel or Windows were specified instead, then the source file would open as a spreadsheet in the Desktop Client instead of as an Axiom form.

```
=GetDocumentHyperlink()
```

This example creates a hyperlink to launch the current document. If you want to specify a client version in this case, you must delimit the omitted first parameter with an "empty" comma, like so: `=GetDocumentHyperlink(,"Excel")`

```
=GetDocumentHyperlink(930,,"GL;Dept.VP='Jones'",,"Report!D22")
```

This example creates a hyperlink to launch the file that has document ID 930. When the file is launched, a sheet filter is applied that restricts the data to VP Jones. The filter applies to all tables in the GL table type. Additionally, the cursor will be placed at cell D22 on the Report sheet.

```
=GetDocumentHyperlink(930,"Windows",,"en-US")
```

This example creates a hyperlink to launch the file that has document ID 930, using the Windows Client. The file will be opened in an English Axiom session, regardless of the regional format used by the current operating system.

GetDocumentInfo function

Returns information about the current file.

▶ Syntax

```
GetDocumentInfo("DocInfo", "VariableName", "DefaultValue")
```

Parameter	Description
DocInfo	A keyword that specifies which piece of information about the document that you want the function to return. See the DocInfo Value table below for the supported list of values.
VariableName	This parameter only applies if the DocInfo value is `Variable`. The variable name for which you want to return a value. The variable name and value are passed to the current file by the GetDocument function.
DefaultValue	This parameter only applies if the DocInfo value is `Variable`. Optional. The default value for the specified variable name. The function will return this default value until a new value is passed to the file using the GetDocument function.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

The following values are supported for the DocInfo parameter:

DocInfo Value	Description
ActiveSheetName	Returns the name of the sheet that was active when the last recalculation occurred.
CurrentSheetName	Returns the name of the current sheet (the sheet where the function resides).
Description	Returns the description of the current file, if defined.
DocumentID	Returns the ID of the current file (a numeric value assigned by Axiom Software).
DocumentName	Returns the file name of the current file.
DocumentType	Returns the document type for the current file (such as Report or Template).
EmbeddedFormType	Returns the embedded context for a form.
FolderID	Returns the ID of the folder where the current file is located (a numeric value assigned by Axiom Software).
FolderName	Returns the name of the folder where the current file is located. Only the folder name is returned, not the full path.
FolderPath	Returns the folder path where the current file is located. The file name is not part of that path.
FullPath	Returns the folder path and file name of the current file. This is the same path that would display if looking up the file in Axiom Explorer.

DocInfo Value	Description
IsReadOnly	Returns whether the file is opened read-only (as True/False).
LockedByUser	Returns the login name of the user who currently has the file lock.
RecordCreatedBy	Returns the login name of the user who created the file.
RecordCreatedDTM	Returns the date and time the file was created.
RecordModifiedBy	Returns the login name of the user who last modified the file.
RecordModifiedDTM	Returns the date and time the file was last modified.
Variable	Returns the value of a variable that was passed by the GetDocument function. For more information on using this option, see Passing values from one file to another.

▶ Remarks

- GetDocumentInfo only applies to managed files in the Axiom Software virtual file system. If used in a non-managed file, the function returns a blank value.
- GetDocumentInfo is a volatile function.

▶ Examples

`=GetDocumentInfo("FolderName")`

This example returns the name of the folder where the file is stored. For example: Budget Reports.

`=GetDocumentInfo("DocumentName")`

This example returns the file name of the file. For example: Income Statement.xlsx.

`=GetDocumentInfo("DocumentID")`

This example returns database ID of the file. For example: 219.

`=GetDocumentInfo("FullPath")`

This example returns the full path of the file. For example: \Axiom\File Groups\2018 Budget\Plan Files\11000.XLSX

`=GetDocumentInfo("IsReadOnly")`

This example returns True if the file is opened read-only. You might use this in an IF function in the header of a file to provide a message to the user, such as "ALERT: This file is read-only; changes cannot be saved."

GetRowNumber function

Returns the row number of the current cell, of the cell that was active when the file was refreshed, or for a specific cell reference.

The intent of GetColumnLetter is to help support custom drilling in Axiom files using double-click.

`GetRowNumber("`*`Cell`*`")`

The parameter Cell takes the following values:

- <Blank>: Leave the Cell parameter blank (open parentheses) to return the row number for the current cell (the cell that contains the GetRowNumber function).
- `Active`: Use this keyword to return the row number for the cell that was active when the last recalculation occurred.
- *CellReference*: Enter a cell reference to return the row number for that cell.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- Drilling initiates a recalculation, so `Active` can be used to identify the row where the user double-clicked to perform the drill.
- GetRowNumber is a volatile function.

▶ Examples

`=GetRowNumber()`

This example returns the row number for the cell that contains the GetRowNumber function (for example: "1").

`=GetRowNumber("Active")`

This example returns the row number for the active cell when the file was refreshed (for example: "20").

`=GetRowNumber("AD25")`

This example returns the row number for the specified cell reference ("25" in this case).

File Group Functions

This section contains information on functions that return information about file groups, such as file group properties and file group variables.

GetFileGroupID function

Returns the ID of the current file group, or of a specified file group (given the file group name). This function should be used to obtain the file group ID for subsequent use in the functions GetFileGroupVariable and GetFileGroupProperty.

▶ Syntax

```
GetFileGroupID("FileGroupName")
```

Parameter	Description
FileGroupName	Optional. The name of the file group for which you want to return the ID. A file group alias name can also be used.
	• If you are using the function within a file that belongs to a file group (template, driver, etc.), then you can omit the name and the current file group is assumed.
	• If you are using the function outside of a file group (such as in a report file), then you must specify a file group name or alias name.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

• When using this function within a file that belongs to a file group, it is best to leave the FileGroupName parameter blank so that you are always returning the ID for the current file group. If you provide the function with the name of the current file group, it is possible that the function will get out of sync with the file group if the name changes (for example, when cloning the file group for a rollover).

- The file group name corresponds to the **File Group Name** property for the file group, not the display name. If a variable is used in the name, you must resolve the variable to its current value. For example, if the name is defined as `Budget {FileGroupYear}`, you must specify `Budget 2019` in the function (or whatever year the variable currently resolves to).

- GetFileGroupID is a non-volatile function.

▶ Examples

`=GetFileGroupID()`

This example returns the ID for the current file group, assuming the function is used within a file that belongs to a file group. For example: 89.

`=GetFileGroupID("Budget 2019")`

This example returns the ID for the specified file group. For example: 89. The file group name should only be specified when the function is used in a report file that does not belong to a file group (or if you are deliberately referencing a file group that is not the current file group).

`=GetFileGroupID("Current Budget")`

This example uses a file group alias name to return the ID for the file group assigned to the alias. When the file group assignment is changed for the alias (such as for a rollover), then the function will return the ID for the newly assigned file group.

GetFileGroupInfo function

The GetFileGroupInfo function has been deprecated in favor of the following functions:

- GetFileGroupID: Returns the ID of a file group.
- GetFileGroupVariable: Returns the value of a file group variable, given the variable name.
- GetFileGroupProperty: Returns the value of various file group-related properties, such as the file group display name, or the current plan code.

If you have existing GetFileGroupInfo functions in your files, the functions will continue to work. However, we strongly recommend converting these deprecated functions to use the new functions instead.

> **NOTE:** The new functions replace all of the functionality of the GetFileGroupInfo function, except for the ability to return the file group code. Use of the file group code to identify a file group has been deprecated. Instead, you should use the file group name or the file group ID to identify a file group.

GetFileGroupProperty function

Returns information about a file group, such as the file group name or the current plan code.

NOTE: You can use a data lookup to return the same information as a GetFileGroupProperty function. Whenever possible, it is recommended to use data lookup queries over Axiom functions for faster performance.

▶ Syntax

`GetFileGroupProperty("`*CodeName*`", `*FileGroupID*`)`

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. • `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.
FileGroupID	Optional. The database ID of the file group. • If you are using the function within a file that belongs to a file group (template, driver, etc.), then you can omit the ID and the current file group is assumed. • If you are using the function outside of a file group (such as in a report file), then you must specify a file group ID. The function GetFileGroupID can be used to return the file group ID as necessary.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

Remarks

- When using this function within a file that belongs to a file group, you may see a small performance improvement if you include the file group ID in the function rather than omitting it to assume the current file group. In this case you should have a single designated cell that uses the GetFileGroupID function to return the ID for the current file group, and then reference that cell to get the ID.

- GetFileGroupProperty is a non-volatile function.

Examples

`=GetFileGroupProperty("FileGroupName")`

This example returns the name of the current file group, assuming the function is used within a file that belongs to a file group. For example: Budget 2019.

`=GetFileGroupProperty("PlanFile",Variables!$F$20)`

This example returns the plan code for the current plan file. For example: 100. In this example the file group ID is being specified by pointing to a designated cell on the Variables sheet that contains a GetFileGroupID function.

`=GetFileGroupProperty("ScenarioName",83)`

This example returns the scenario name of the file group with ID 83. For example: V1. In a report file, the file group ID must be specified because there is no current file group context. In most cases the ID would not be "hard-coded" in the function as shown here—instead it would be returned via the GetFileGroupID function or from a query to a system table such as Axiom.FileGroups.

`=GetFileGroupProperty("UtilitiesPath")`

This example returns the path to the Utilities folder for the current file group. For example: `\Axiom\SystemFolderName_FileGroups\Capital Requests\SystemFolderName_Utilities`. This could be used to build up dynamic paths to certain driver files, so that the paths would automatically update when the file group is cloned.

GetFileGroupVariable function

Returns the value for a file group variable, given the variable name.

> **NOTE:** You can use a data lookup to return the same information as a GetFileGroupVariable function. Whenever possible, it is recommended to use data lookup queries over Axiom functions for faster performance.

Syntax

`GetFileGroupVariable("`*`VariableName`*`",`*`FileGroupID`*`)`

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.
FileGroupID	Optional. The database ID of the file group.
	• If you are using the function within a file that belongs to a file group (template, driver, etc.), then you can omit the ID and the current file group is assumed.
	• If you are using the function outside of a file group (such as in a report file), then you must specify a file group ID. The function GetFileGroupID can be used to return the file group ID as necessary.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- When using this function within a file that belongs to a file group, you may see a small performance improvement if you include the file group ID in the function rather than omitting it to assume the current file group. In this case you should have a single designated cell that uses the GetFileGroupID function to return the ID for the current file group, and then reference that cell to get the ID.

- GetFileGroupVariable is a non-volatile function.

▶ Examples

`=GetFileGroupVariable("FileGroupYear")`

This example returns the value of the built-in FileGroupYear variable, for the current file group. For example: 2019.

`=GetFileGroupVariable("BudgetData",Variables!$F$32)`

This example returns the value of the user-defined BudgetData table variable, as defined in the file group properties of the current file group. For example: BGT2019. In this example the file group ID is being specified by pointing to a designated cell on the Variables sheet that contains a GetFileGroupID function.

`=GetFileGroupVariable("BudgetData",83)`

This example returns the value of the BudgetData table variable, for the file group with ID 83. In a report file, the file group ID must be specified because there is no current file group context. In most cases the ID would not be "hard-coded" in the function as shown here—instead it would be returned via the GetFileGroupID function or from a query to a system table such as Axiom.FileGroups.

`=GetFileGroupVariable("SalaryEscalator")`

This example returns the value of the user-defined SalaryEscalator variable, as defined in the file group properties of the current file group. For example: .02. If the variable is defined as numeric then the value

will be returned as a number; otherwise it will be returned as a string.

GetFileGroupVariableEnablement function

Returns whether a picklist variable is enabled for a specified value in the picklist enablement column (True/False).

▶ Syntax

`GetFileGroupVariableEnablement("`*VariableName*`", `*EnablementValue*`, `*FileGroupID*`)`

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.
FileGroupID	Optional. The database ID of the file group. • If you are using the function within a file that belongs to a file group (template, driver, etc.), then you can omit the ID and the current file group is assumed. • If you are using the function outside of a file group (such as in a report file), then you must specify a file group ID. The function GetFileGroupID can be used to return the file group ID as necessary.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ 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.

- When using GetFileGroupVariableEnablement within a file that belongs to a file group, you may see a small performance improvement if you include the file group ID in the function rather than omitting it to assume the current file group. In this case you should have a single designated cell that uses the GetFileGroupID function to return the ID for the current file group, and then reference that cell to get the ID.

- GetFileGroupVariableEnablement is a non-volatile function.

▶ Examples

```
=GetFileGroupVariableEnablement("Category", 1)
```

This example returns True if the Category variable is enabled for value 1, and False if it is not. The function is used within a file that belongs to a file group, so the file group ID is omitted.

```
=GetFileGroupVariableEnablement("Category", Variables!$F$27, Variables!$F$10)
```

This example is the same as the previous example, except that the enablement value and the file group ID are being read from another sheet using a cell reference. In most cases these values will be queried into the file using a GetData data lookup or function.

GetFileGroupVariableProperty function

Returns information about file group variable properties, given the name of a table variable or a picklist variable.

> **NOTE:** This function does not return the value of a variable. For that, use GetFileGroupVariable.

▶ Syntax

```
GetFileGroupVariableProperty("VariableName", "CodeName", FileGroupID)
```

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.
FileGroupID	Optional. The database ID of the file group.
	• If you are using the function within a file that belongs to a file group (template, driver, etc.), then you can omit the ID and the current file group is assumed.
	• If you are using the function outside of a file group (such as in a report file), then you must specify a file group ID. The function GetFileGroupID can be used to return the file group ID as necessary.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- When using this function within a file that belongs to a file group, you may see a small performance improvement if you include the file group ID in the function rather than omitting it to assume the current file group. In this case you should have a single designated cell that uses the GetFileGroupID function to return the ID for the current file group, and then reference that cell to get the ID.

- All variable properties returned by this function are defined for the variable in the **Edit File Group** dialog, on the **Variables** tab.

- GetFileGroupVariableProperty is a non-volatile function.

`=GetFileGroupVariableProperty("Category", "ColumnName")`

This example returns the name of the column that is associated with the picklist variable Category. For example: CPREQ2018.Category. The function is used within a file that belongs to a file group, so the file group ID is omitted.

`=GetFileGroupVariableProperty("Category", "EnabledValues", Variables!$F$20)`

This example returns the list of enabled values associated with the picklist variable Category. For example: 1,4. In this example the file group ID is being specified by pointing to a designated cell on the Variables sheet that contains a GetFileGroupID function.

`=GetFileGroupVariableProperty("GLData", "IsReadOnly")`

This example returns whether the table variable GLData is flagged as read-only. For example: True. This means that the file group does not save data to the table that GLData resolves to.

`=GetFileGroupVariableProperty("Category", "IsRequired")`

This example returns whether the picklist variable Category is flagged as required. For example: True. You could use this information to require the user to select a category for the plan file, by enabling custom save validation for a save-to-database process.

GetPlanFileAttachment function

Opens the Browse Attachments dialog so that a user can open a plan file attachment, or opens a specified plan file attachment directly. This function can be used in an Axiom file to provide an alternate method of opening plan file attachments.

The function can also be used to write the name of a selected plan file attachment to a specified target cell.

> **NOTE:** GetPlanFileAttachment is not for use in Axiom forms. If you want to open plan file attachments from within an Axiom form, you must use either GetFormResourceLinkTag or GetFormResourceURL. For more information, see the *Axiom Forms and Dashboards Guide*.

▶ Syntax

`GetPlanFileAttachment("DisplayText", "FileGroupName", PlanFileCode, OpenReadOnly, "AttachmentNameorTargetCell")`

Parameter	Description
DisplayText	The text to display in the cell.
	This text displays as normal text in the cell. If you want the text to appear like a hyperlink on a web page, you must manually apply the font formatting to the cell.
	This parameter supports use of the variable `{count}`, which can be used to display the count of attachments for the plan file.
FileGroupName	The name of the file group that the plan file belongs to. A file group alias name can also be used.
PlanFileCode	The plan file code for which you want to open an associated file attachment, such as 21000.
OpenReadOnly	Optional. Boolean value to determine whether the file is opened read-only.
	By default, this is `FALSE`, which means that the file will open with the user's normal level of access rights, as determined by their access rights to the associated plan file. If `TRUE`, the file is opened as read-only, regardless of the user's access rights.
	This option only applies to Excel-compatible spreadsheet attachments that are opened within the Axiom Software session. It is ignored for all other types of attachments because those attachments cannot be edited directly within Axiom Software.
AttachmentNameor TargetCell	Optional. If this parameter is omitted, then the **Browse Attachments** dialog opens so that the user can select an attachment to open.
	This parameter can be used to enable special behavior for the function as follows:
	• Specify the file name of the specific attachment to be opened. In this case, the Browse Attachments dialog does not open and instead the specified attachment opens directly.
	• Specify a target cell to write the name of a selected attachment. The target cell must be placed in quotation marks, such as "E24", so that it is not interpreted as a cell reference. When using a target cell, the Browse Attachments dialog opens so that the user can select an attachment. However, instead of opening the attachment, the attachment name is written to the specified target cell.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell. However, the Boolean values TRUE and FALSE do not need to be in quotation marks.

▶ Remarks

- The second parameter of this function also accepts a file group code. However, use of file group code is being phased out. If you have existing functions that use the file group code, these functions should eventually be migrated to use the file group name.

- Users cannot manage plan file attachments using this function, they can only open existing plan file attachments. If users need to add or delete plan file attachments, they must open the plan file and use the **File Attachments** command on the ribbon, or use the ManagePlanFileAttachments function.

- GetPlanFileAttachment is a non-volatile function. This means that if the `{count}` variable is used, it will not automatically update when the count changes—the sheet must be refreshed in order to update the function.

▶ Examples

`=GetPlanFileAttachment("Open Attachments for Dept 21000","Budget 2019",21000)`

This example opens the Browse Attachments dialog so that the user can open any attachment for plan file 21000 in file group Budget 2019.

`=GetPlanFileAttachment("Open supporting schedule","Budget 2019",21000,,"Schedule.xlsx")`

This example opens the specified file attachment for plan file 21000 in file group Budget 2019. The attachment is an Excel file so it will be opened in the current Axiom Software session, using the level of access rights that the user has for plan file 21000 (either read-only or read/write).

`=GetPlanFileAttachment("Open supporting schedule","Budget 2019",21000,TRUE,"Schedule.xlsx")`

This example is the same as the previous example, except this time the Excel file will be opened as read-only, regardless of the user's access rights to the plan file.

`=GetPlanFileAttachment("Open proposal","Budget 2019",21000,,"Proposal.docx")`

This example opens the specified file attachment for plan file 21000 in file group Budget 2019. The attachment is a Word file so it will open in Word, assuming that program is present on the user's computer.

`=GetPlanFileAttachment("{count} attachments","Budget 2019",21000)`

This example uses the `{count}` variable to display the count of attachments for the specified plan file within the display text. For example, the display text will read as "3 attachments".

`=GetPlanFileAttachment("Select attachment","Budget 2019",21000,,"G27")`

This example uses a target cell for the fifth parameter. This example opens the Browse Attachments dialog so that the user can view the attachments for plan file 21000 in Budget 2019. When the user selects an attachment, the attachment name is written to the target cell instead of opening that attachment, such as "Schedule.xlsx".

GetPlanFileDocumentID function

Returns the document ID for a particular plan file, given a file group name and a plan code.

> **NOTE:** Whenever possible, you should use the function GetPlanFilePath instead of GetPlanFileDocumentID, for improved performance. Most settings that use a document ID can use a path instead.

▶ Syntax

GetPlanFileDocumentID("*FileGroupName*", *PlanCode*)

Parameter	Description
FileGroupName	The name of the file group. A file group alias name can also be used.
PlanCode	The plan code for which to return the document ID.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- The first parameter of this function also accepts a file group code. However, use of file group code is being phased out. If you have existing functions that use the file group code, these functions should eventually be migrated to use the file group name.

- GetPlanFileDocumentID is a non-volatile function.

▶ Examples

=GetPlanFileDocumentID("Budget 2019",3000)

This example returns the document ID for the plan file for department 3000, for file group Budget 2019. For example: 93.

GetPlanFilePath function

Returns the file path of a particular plan file, given a file group name and a plan code.

▶ Syntax

GetPlanFilePath("*FileGroupName*", *PlanCode*)

Parameter	Description
FileGroupName	The name of the file group. A file group alias name can also be used.
PlanCode	The plan code for which to return the file path.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

GetPlanFilePath is a non-volatile function.

▶ Examples

`=GetPlanFilePath("Budget 2019", 3000)`

This example returns the file path of the plan file for department 3000, for file group Budget 2019. For example: `\Axiom\SystemFolderName_FileGroups\Budget 2017\SystemFolderName_Plans\3000`.

You might use this function to provide a file path to another Axiom function, for example:

```
=GetFormDocumentLinkTag("Open plan file for Dept 3000", GetPlanFilePath
("Budget 2019", 3000), , TRUE)
```

File Processing Functions

This section contains information on functions for use with file processing. These functions can be used to return information about the current pass, and to indicate whether multipass processing is occurring.

GetCurrentValue function

Returns information about the current filter context applied to the file, for either multipass processing or for a temporary sheet filter.

Temporary filters supported by this function include:

- Sheet filters applied by use of the Quick Filter feature.
- Sheet filters applied by use of a GetDocumentHyperlink URL.

This function is most commonly used for report titles and headers, so that the information updates dynamically when different filters are applied to the file.

▶ Syntax

```
GetCurrentValue("ColumnNameorCode")
```

The valid entries for *ColumnNameorCode* depends on whether you are using the function in conjunction with multipass processing or with a temporary sheet filter.

Multipass processing

Value	Description
Blank	If the parameter is left blank (open parentheses), the function returns the item being processed for the current pass. For example, if the multipass process is by department, then the function returns the department code for the current pass.

Value	Description
ColumnName	You can specify any column listed as a **Source Column** for the multipass process, and the function returns the value in that column, for the current pass. You can also specify columns that are listed only in the **Group By** and **Sort By** settings for the multipass process, but in this case no values can be returned for standard (non-multipass) processing. See GetCurrentValue function for more information. For example, if the multipass process is by department, and you specify DEPT.VP as the column name, then the function returns the VP of the department for the current pass. You can use fully qualified Table.Column syntax to specify the column, or you can use column-only syntax.
PassNumber	You can enter the keyword `PassNumber` to return the number of the current pass in the context of the overall multipass operation. For example, if the multipass process has 10 items to process, and the current item is the fifth item to be processed, the function returns 5.
Filter	You can enter the keyword `Filter` to return the full filter criteria statement being applied for the current pass. For example, this would return Dept.Region='West' when processing by Dept.Region and the current pass is for the West region.
Table	You can enter the keyword `Table` to return the table being used for the multipass process. For example, this would return Dept when processing by Dept.Region.
Column	You can enter the keyword `Column` to return the column being used to determine the list of values for the multipass process. For example, this would return Region when processing by Region. When using the Column keyword, an optional Boolean parameter is available to specify whether to return just the column name or the fully qualified column name. By default, this parameter is False, which means the column name is returned. To return the fully qualified column name, specify True. For example: `GetCurrentValue("Column",True)` to return Dept.Region instead of just Region.

Temporary sheet filter

Value	Description
QuickFilter	You can enter the keyword `QuickFilter` to return the filter criteria statement that is being applied by a temporary sheet filter. For example, if the sheet filter uses the filter criteria statement `Dept.VP='Jones'`, that value is returned. If no temporary sheet filter is currently applied to the file, the function returns `None`. **NOTE:** The prior version of this keyword is `Temp_Filter_Value`.
Temp_Filter_Key	You can enter the keyword `Temp_Filter_Key` to return the table or table type that is being used as the "key" for a temporary sheet filter. For example, if the filter uses the table type GL, that table type name is returned. If no temporary sheet filter is currently applied to the file, the function returns `None`.

All non-numeric entries must be placed in double quotation marks, unless you are using cell references to reference the text held in another cell.

▶ Remarks

- The return values for multipass processing and temporary sheet filters are mutually exclusive. The codes used for multipass processing do not recognize temporary sheet filters, and the codes used for temporary sheet filters do not recognize multipass processing. The function cannot be constructed in a way to return a filter value in both contexts.
- When using the file processing type of Save Data in Batches, the only valid use of GetCurrentValue is to return the pass number. The other options do not apply in this context and will not return data.
- GetCurrentValue is a non-volatile function.

▶ Multipass examples

`=GetCurrentValue()`

This example returns the value of the current item being processed. For example, "Jones" if processing by DEPT.VP.

`=GetCurrentValue("DEPT.VP")`

This example returns the value in the DEPT.VP column for the current item being processed.

The difference between this example and the first one is that DEPT.VP does not have to be the column that defines the multipass list of items. For example, you may be processing by departments, but you have included DEPT.VP as an additional source column. As each department is processed, this example returns the name of the VP associated with that department.

Note that you could accomplish the same result by passing the current item value into GetData: `=GetData("vp","dept="&GetCurrentValue()&"","dept")`

```
=GetCurrentValue("PassNumber")
```

This example returns the number of the current pass. For example, "5" if the current pass is the fifth pass of the process. If the multipass process has 10 passes, this function will return 1-10 for each successive pass.

```
=GetCurrentValue("Filter")
```

This example returns the filter being applied for the current pass. For example, "Dept.Region='West'".

▶ Temporary sheet filter examples

```
=GetCurrentValue("QuickFilter")
```

This example returns the filter criteria statement of the temporary sheet filter applied by the Quick Filter feature or GetDocumentHyperlink. For example: `DEPT.Region='West'`.

```
=GetCurrentValue("Temp_Filter_Key")
```

This example returns the table or table type that the temporary sheet filter applies to. For example: `DEPT`.

▶ GetCurrentValue behavior during non-multipass processing

Although the primary intent of the GetCurrentValue function is to return information during multipass processing, you can configure the multipass settings so that the function also returns relevant values when using non-multipass processing (**Process File**). This applies when using GetCurrentValue with open parentheses or with a source column name.

The top of the File Processing Control Sheet has a section to define source columns and their current value defaults. For each source column listed in the top row, you can define a corresponding current value default in the bottom row. For example:

Multipass Columns and Current Value Defaults	
Source Columns	Dept.VP
Current Value Defaults	All VPs

In this example, when using multipass processing, `GetCurrentValue("DEPT.VP")` returns the name of the VP currently being processed. When using non-multipass processing, the same function returns the text "All VPs". If the default value was left blank, then the function would return nothing (blank) during non-multipass processing.

You can define a default value for each source column as desired, or leave it blank to return blank.

If you use the GetCurrentValue function with open parentheses—`GetCurrentValue()`—then a value will only be returned during non-multipass processing if the **Group By** column is also listed as a source column (this happens automatically when using basic mode multipass settings), and the source column has a default value defined on the control sheet. Otherwise, the open parentheses function returns nothing during non-multipass processing.

When no file processing is occurring, using open parentheses or a column name returns the default values if defined; otherwise the function returns blank.

> **NOTES:**
> - You must use the File Processing Control Sheet if you want to define current value defaults for your source columns. The File Processing pane does not have a section to define these values.
>
> - The other options for GetCurrentValue are only intended to return values during multipass processing. This applies to the keywords PassNumber, Filter, Table, and Column. All of these options return blank when multipass processing is not occurring, except for PassNumber. PassNumber returns 0 if no processing is occurring, and 1 when non-multipass processing is occurring.

IsRunningMultiPass function

Returns True if the file that contains the function is currently being processed using multipass processing. Can be used to create dynamic headers or change file settings based on whether multipass processing is occurring.

▶ Syntax

```
IsRunningMultiPass()
```

This function does not take any parameters.

▶ Remarks

IsRunningMultiPass is a non-volatile function.

▶ Examples

```
=IsRunningMultiPass()
```

This example returns "False" when multipass processing is not occurring, and "True" when it is occurring.

```
=IF(IsRunningMultipass()=TRUE,GetCurrentValue(),"Consolidated")
```

In this example, the IsRunningMultiPass function is used within an IF function to change the result depending on whether multipass processing is occurring. This function returns the current pass item during multipass processing, and returns the text "Consolidated" otherwise. (You can get the same end result by using just the GetCurrentValue function, if you define the current value default as "Consolidated" for the column that defines the multipass list of items.)

Supporting Information

Passing values from one file to another

When opening one Axiom file from another Axiom file, you can pass values to the target file using document variables. For example, you may want to set certain values in the first file, then pass those values to the target file to impact the data queries in the target file. This applies to the following functions:

- GetDocument
- GetFormDocumentURL
- GetFormDocumentLinkTag

The GetDocument function is used when you want to open a new spreadsheet Axiom file from within another spreadsheet Axiom file. In this case, the user double-clicks the function in the starting file to open the target file.

The other two functions are used when you want to open a new Axiom form from within another Axiom form.

- The GetFormDocumentURL function can be used to generate a URL to another Axiom form, and then that URL can be used in a Hyperlink component or in an HREF tag for a Formatted Grid component.
- The GetFormDocumentLinkTag function can be used to generate a fully constructed HREF tag for a Formatted Grid component. The user clicks the resulting hyperlink in the starting form to open the target form.

The ability to pass values to the target file is set up as follows:

- In the starting file, use the Variables parameter of the function to pass variable/value pairs to the target file.
- In the target file, use the GetDocumentInfo function to return the value for each variable that was passed to the file. You can then reference these values to impact the data query.

For example, imagine that you want to determine a region within the starting file. When the function is used to open the target file, you want to pass the region to the target file and filter the data queries to show only data for that region. The following discussion uses this example to illustrate how to set up this

relationship between files. This example uses the GetDocument function, but the same principles apply to the other two functions.

▶ Setting up the starting file

In the starting file, you must define the document variable and the value that you want to pass to the target file. In this example, the variable/value pair is as follows:

```
Region=Value
```

Where Region is the variable and *Value* is the specific region to pass to the target file.

You can use any method that you want to determine the region value. For example, you might use formulas to automatically determine the appropriate region. Or you might ask the user to select a region from a drop-down list.

Once the region value is determined, you must include the variable/value pair in the GetDocument function. For example, imagine that the region value is stored in cell B2 of the Variable sheet. The GetDocument function might look as follows:

```
=GetDocument("Open Report", 93, "Report",,,,,,, "Region="&Variable!B2)
```

This example looks to cell B2 of the Variable sheet to pick up the defined value for Region, which is US West.

You can include multiple variable/value pairs in the Variables parameter, separated by semicolons. For example, if we wanted to also pass a VP name, the GetDocument function might look as follows:

```
=GetDocument("Open Report", 93, "Report",,,,,,
"Region="&Variable!B2&";VP="&Variable!B3)
```

When the user double-clicks on the GetDocument function, the target file is opened and the variable values are passed to that file.

▶ Setting up the target file

In the target file, you must use the GetDocumentInfo function to return the variable values that were passed to the file by the GetDocument function. For example, to return the Region value you would use a function like the following:

```
=GetDocumentInfo("Variable","Region")
```

The first parameter tells Axiom Software that you want to return the value of a passed variable. The second parameter specifies the variable value to return. In this example, the function returns the value for the Region variable, which is US West.

You would set up the queries in this file so that they are filtered by the value returned by the GetDocumentInfo function. In most cases, you would also want to enable "refresh on open" (so that the queries are run when the file is opened and the filter is applied), and set up dynamic headers that reference the region value.

Keep in mind that until a variable value is passed to the file using the GetDocument function, the GetDocumentInfo function will return blank. If you need the queries in this file to work even when no variable value has been passed, you have two options:

- You can set up the formula for the query filter so that a different filter (or no filter) is used if the GetDocumentInfo function returns blank.
- You can define a default value for the variable within the GetDocumentInfo function. The function will then return that default value until a new value is passed using the GetDocument function.

The default value is the third parameter of the GetDocumentInfo function. For example, a default region value could be set within the function as follows:

```
=GetDocumentInfo("Variable","Region","US Central")
```

This function will now return the value US Central when no value is passed using the GetDocument function.

Volatile and non-volatile functions

Functions in Microsoft Excel are either volatile or non-volatile. Cells that contain volatile functions are evaluated every time a file is recalculated, as well as any dependent cells, regardless of whether any of the function's arguments have changed. Non-volatile functions are evaluated only when the function's arguments have changed.

Use of many volatile functions in a file can dramatically slow down calculation times for that file. Whenever possible, you should use non-volatile design alternatives and limit use of volatile functions.

Be aware that if a non-volatile function references a volatile function in its parameters, then the non-volatile function will behave as if it is volatile.

Microsoft Excel functions

The following Microsoft Excel functions are volatile (view source MSDN document):

- Now
- Today
- Rand
- Offset
- Indirect
- Info (depending on its arguments)
- Cell (depending on its arguments)

Axiom functions

The following Axiom functions are volatile:

- GetColumnLetter
- GetCurrentSheetView
- GetFormState
- GetDocumentInfo
- GetRowNumber
- GetSharedVariable
- GetUserInfo
- RunEvent
- SetSharedVariable

Using cell references in Axiom functions

Instead of placing values directly into Axiom functions, you can place the values elsewhere in the spreadsheet and then use cell references. Use of cell references allows you to create more dynamic functions—by changing one or two values in a sheet, you can change many or all of the functions throughout the sheet.

For example, consider the following formula:

```
=GetData("NYB1","ACCT.AcctGroup='Benefits'","Plan2019")
```

Instead of stating the column and filter within the function, you could place those values elsewhere and use a formula like the following:

```
=GetData(E5,C10,"Plan2019")
```

When using cell references, the cell references are not placed in quotation marks, and the text in the referenced cells is not placed in quotation marks. The exception is the filter criteria—if the item referenced is text, it must be placed in single quotation marks within the referenced cell (such as `ACCT.AcctGroup='Benefits'`).

▶ Absolute cell references versus relative cell references

If the information being referenced is always in the same cell, then you should use dollar signs to "lock" the cell reference. When a reference is locked, then the formula will always point to the same cell, regardless of where the formula is copied into the sheet.

For example, if you use $E$5 as the cell reference, then the formula will always point to E5, even if you copy the formula to the next row down or the next column over. This is an absolute cell reference, where the reference always refers to the same cell.

But if you use just E5 as the cell reference, then when you copy the formula to the next row down, it will update to E6. And if you copy the formula to the next column over, it will update to F5. This is a relative

cell reference, where the reference adjusts based on the relative position of the cell that contains the formula and the cell the reference refers to.

You can also use mixed cell references, where just the column or just the row is locked. If you lock just the column ($E5), then the column will remain at column E but the row will adjust. And if you lock just the row (E$5), then the row will remain the same but the column will adjust.

When using cell references in calc methods that will be copied throughout the sheet, make sure to use the appropriate type of cell reference so that the reference will adjust as needed or remain fixed as needed.

▶ Using concatenation and embedded text

If part of the text used in a function is always the same, you can embed it within the function instead of repeating it in each referenced cell. For example, if the first part of each filter criteria statement is the same (ACCT.AcctGroup=), you can embed that text within the function, and place only the end of the criteria statement in the referenced cell (for example, Benefits, Salaries, etc.).

In this example, the function would look like the following:

```
=GetData(E$5,"ACCT.AcctGroup='"&$C10&"'",$F$4)
```

The single quotation marks are necessary so that the end result is `ACCT.AcctGroup='Benefits'` (where Benefits is in cell C10). The double quotation marks around the `&$C10&` are necessary for the function to resolve the value as a string, and the use of ampersands is standard Excel functionality to concatenate the string.

▶ Function parameters that take cell addresses

Some function parameters take cell addresses as their designated value. For example, when using GetDataElement, you can specify a cell address in which to place the selected value.

When specifying a cell address as a parameter value, you must place the cell address in quotation marks like any other text value. For example:

```
=GetDataElement("Select an account","ACCT.Acct","B20")
```

This example places the user's selected account in cell B20. If B20 is *not* in quotation marks, then Axiom Software will use B20 as a cell reference instead of as the designated parameter value. This will still work if cell B20 contains a cell address (or is blank).

Filter criteria syntax

Several areas of Axiom Software 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 =, >,<,<>,<=,>=. 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 Software 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 Software 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 Software 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.

Index