Excel to Graphviz Relationship Visualizer

Turn your Excel data into clear, professional relationship diagrams with Graphviz

Download Now | How it Works

Foundations

Essential concepts and workflow for creating Graphviz diagrams from Excel

Learn the basics →

Apply Style

Fast, expressive graph styling with a built-in style designer and CSS-like styles gallery

Explore styling →

Publish Graphs

Export as image, PDF, SVG, and add animation to SVGs

See publishing options →

Manipulate Data Using SQL

Use SQL to retrieve and filter data from Excel and Access (Windows)

Learn SQL support →

View Graphviz Source

Inspect and export the raw DOT source code

View DOT source →

Exchange Data Using JSON

Version-control friendly JSON format for workbook data

Learn JSON exchange →

Turn Spreadsheets into Beautiful Graphviz Diagrams

The **Relationship Visualizer** spreadsheet transforms your Excel tables into clear, professional **Graphviz diagrams** in seconds. Say goodbye to manual drawing tools — simply enter your data as rows (e.g., "A is related to B"), and watch graphs appear automatically.

Why Users Love It

Whether you're mapping data flows, organization charts, process workflows, timelines, or dependency diagrams, the Relationship Visualizer turns complex relationships into clear, expressive diagrams.

Draws as you type	Live Graphviz rendering as data changes

Powerful styling	Colors, shapes, fonts, arrows, and reusable CSS-like styles
Advanced features	SQL queries, SVG animation, DOT preview, and JSON exchange
Cross-platform	Works on Windows and macOS
Sleek UI	Custom Excel ribbon tabs across all worksheets
Multilingual	English · French · German · Italian · Polish
Absolutely Free	Free to use · No license required
Open Source	MIT License
Rich Code Documentation	AI-powered DeepWiki
Award Winning	SourceForge Community Choice Award
Show Your Support	Buy Me a Coffee

Excel to Graphviz

Copyright

Author

Written and published by **Jeffrey J. Long** ↗

Contact the author at Relationship.Visualizer@gmail.com ↗

Repositories

Description	URL
`.zip` Download Files	https://sourceforge.net/projects/relationship-visualizer/ ↗
Source Assets	https://github.com/jjlong150/ExcelToGraphviz/ ↗
GitHub Pages URL	https://jjlong150.github.io/ExcelToGraphviz/ ↗
Domain Name Alias	https://exceltographviz.com/ ↗

License

This program is free software; you can redistribute it and/or modify it under the terms of the MIT License ↗.

MIT LICENSE

Navigating This PDF

This file is a PDF print of https://exceltographviz.com ↗, created for offline use and long-term reference.

This PDF does not include a traditional Table of Contents. To navigate the document, use your PDF viewer's **Bookmarks** or **Document Outline** panel.

Most PDF readers (Adobe Acrobat, macOS Preview, Chrome, Edge, Firefox) automatically generate a sidebar outline from the site's headings. This provides:

- A hierarchical view of all major sections
- One-click navigation to any topic
- A clear overview of the document's structure

If the bookmarks panel is hidden, look for an icon labeled **Bookmarks**, **Outline**, or **Table of Contents** in your viewer's sidebar controls.

Why no clickable Table of Contents?

Due to limitations in the website-to-PDF conversion process, internal section links cannot be generated reliably across multiple HTML pages. PDF viewer **Bookmarks** provide the most accurate and consistent navigation experience.

Overview

See how the **Relationship Visualizer** turns raw data into clean, meaningful graph diagrams, with styles, views, and publishing built into the workflow.

This quick 2-minute video ↗ gives a feel for what's possible.

Ready to explore further?

Use the cards below to jump straight to examples by topic.

Building the Graph

Get data onto the page and shape the layout.

- Enter Data
- Display Graphs
- Choose Graph Options
- Cluster Elements

Styling

Apply consistent styles and control visibility.

- Design Styles
- Save Styles
- Apply Styles
- Split Data into Views

Publishing

Export a finished graph to share.

- Publish Graphs
- Enhance SVG Output

Data Exchange

Move data in and out using SQL or JSON.

- SQL Data Import
- JSON File Export/Import

Learning & Reference

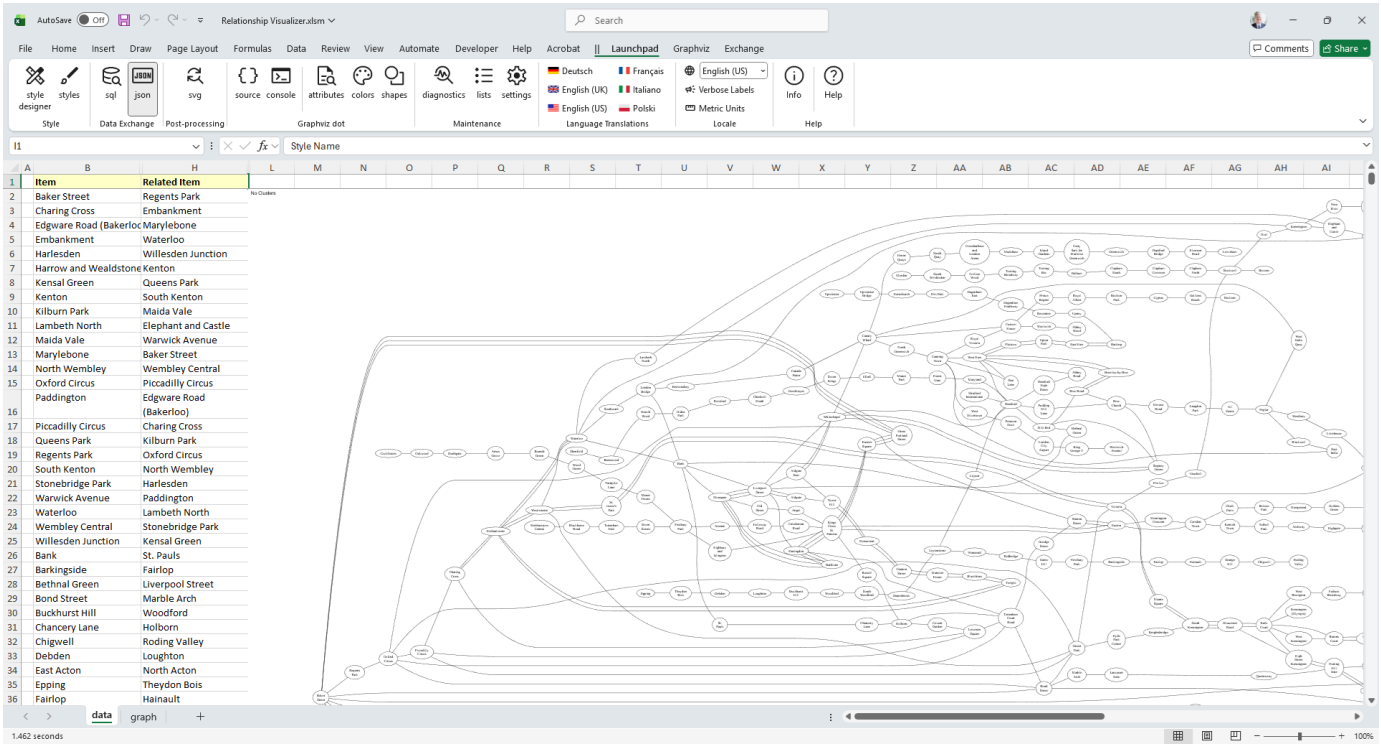
Learn `dot`, troubleshoot, and tweak settings.

- Learn the `dot` Language
- Diagnose Problems
- Internationalization
- Information and Acknowledgements

Enter Data

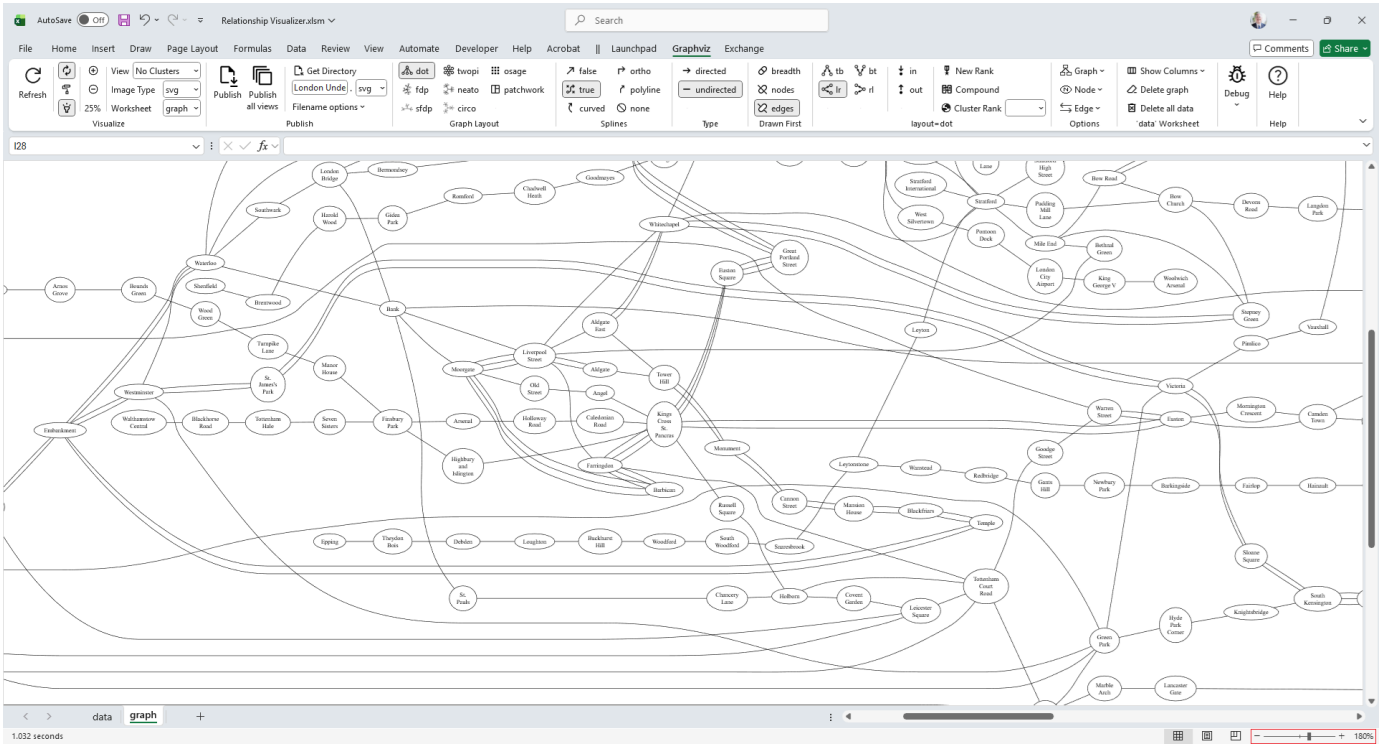
The `data` worksheet is where you specify the relationships to be graphed.

Large data-driven graphs can be constructed easily by simply supplying two columns of data, as shown below:



Display Graphs

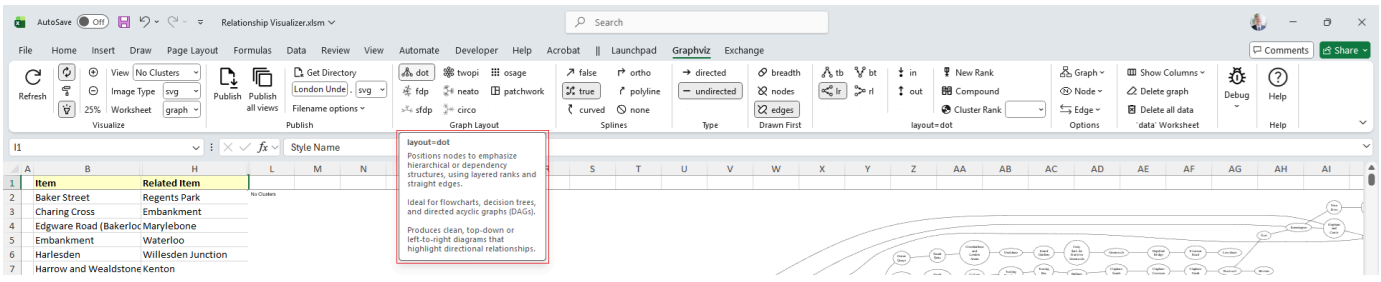
The Relationship Visualizer leverages Excel's zoom in/zoom out, and scrolling capabilities to display graphs in a `graph` worksheet.



Choose Graph Options

The Relationship Visualizer enhances Excel's menu bar and provides all its User Interface controls through Excel's Fluent UI Ribbon tabs. Each control offers robust screen tips, providing helpful information when you hover the mouse over it.

Here is the screen tip displayed when the mouse hovers over the `dot` layout control.

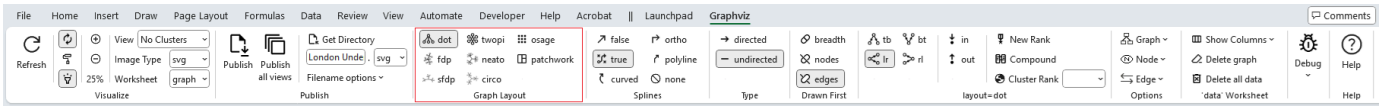


Changing Graphviz options in the Ribbon Tab allows for different Graphviz interpretations of the data. Let's explore examples of *Layout*, *Graph Direction* and *Edge Appearance*.

Graph Layout

The `layout` attribute selects the Graphviz engine that arranges your graph. Different engines use different algorithms, producing distinct structures and visual styles.

Graph layout can be selected with a single button click.



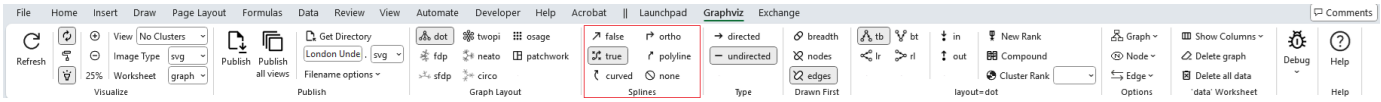
Choices include:

Button	`layout=`	Button	`layout=`	Button	`layout=`
	dot		twopi		osage
	fdp		neato		patchwork
	sfdp		circo		

Edge Appearance

The `splines` attribute controls how Graphviz draws edges between nodes. It determines whether connections appear as smooth curves, straight segments, orthogonal lines, or are omitted entirely.

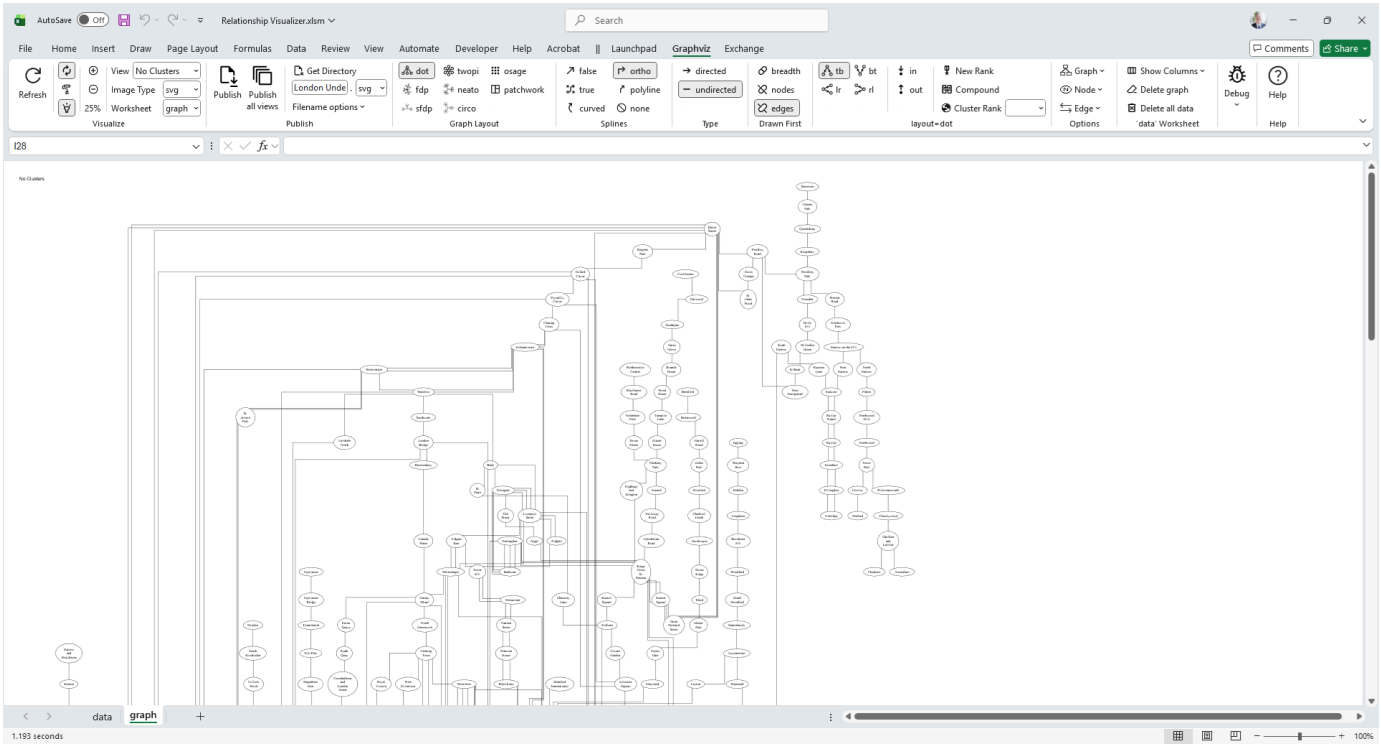
You select how edges are drawn using the `Splines` buttons.



Choices include:

Button	`splines=`	Button	`splines=`
	false (alias=line)		ortho
	true (alias=spline)		polyline
	curved		none

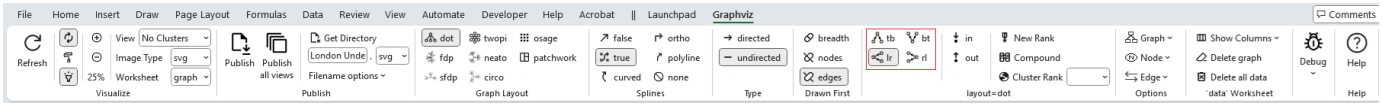
This images shows the previous graph with the edges changed from `true`, to `ortho`, which causes straight lines with 90 degree corners.



Graph Direction

The `rankdir` attribute controls the overall direction in which the graph is laid out. It tells Graphviz whether to arrange ranks from top to bottom, bottom to top, left to right, or right to left. Changing this setting rotates the diagram's flow without altering the underlying structure of the graph.

Graph direction can be selected with a single button click.

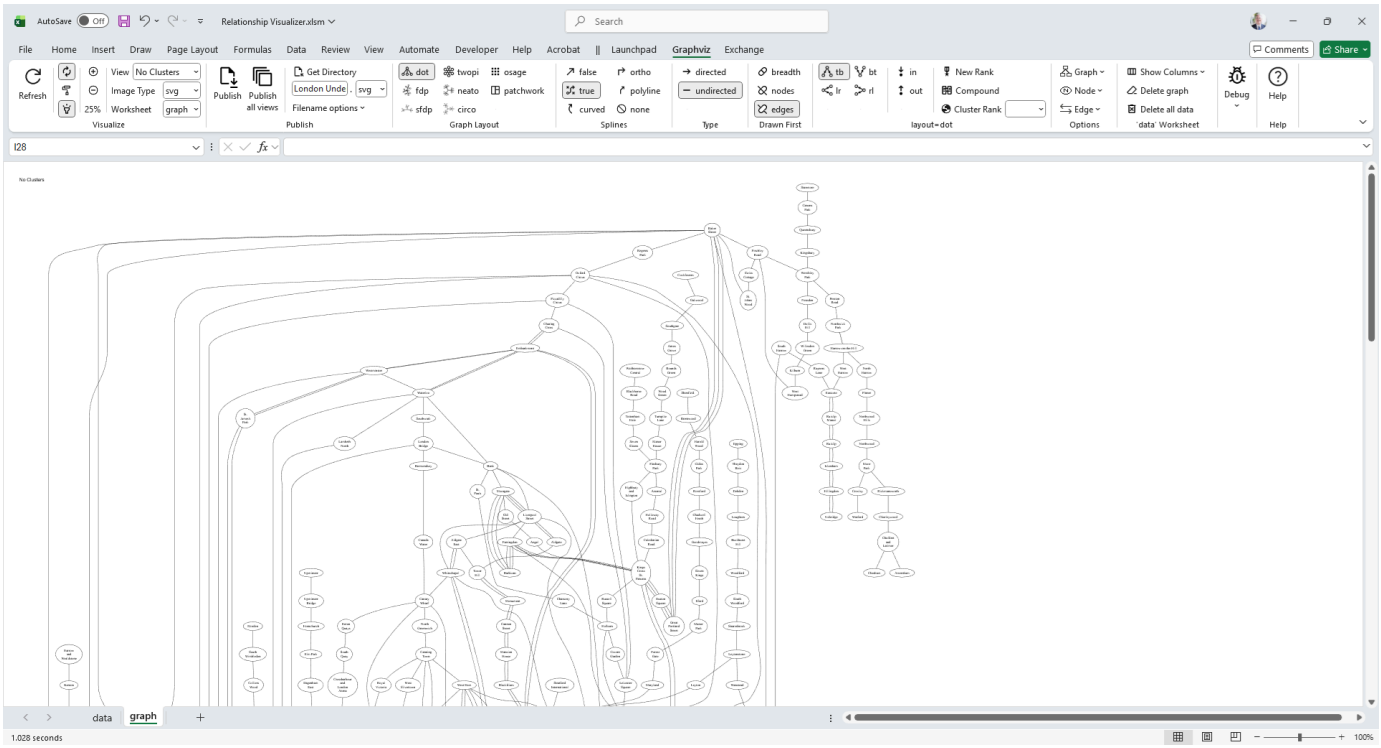


Choices include:

Button	`rankdir=`	Direction
	`tb`	Top to Bottom
	`bt`	Bottom to Top

Button	`rankdir=`	Direction
	`lr`	Left to Right
	`rl`	Right to Left

Here we see the direction of the previous graph has changed from *Left to Right*, to *Top to Bottom*.

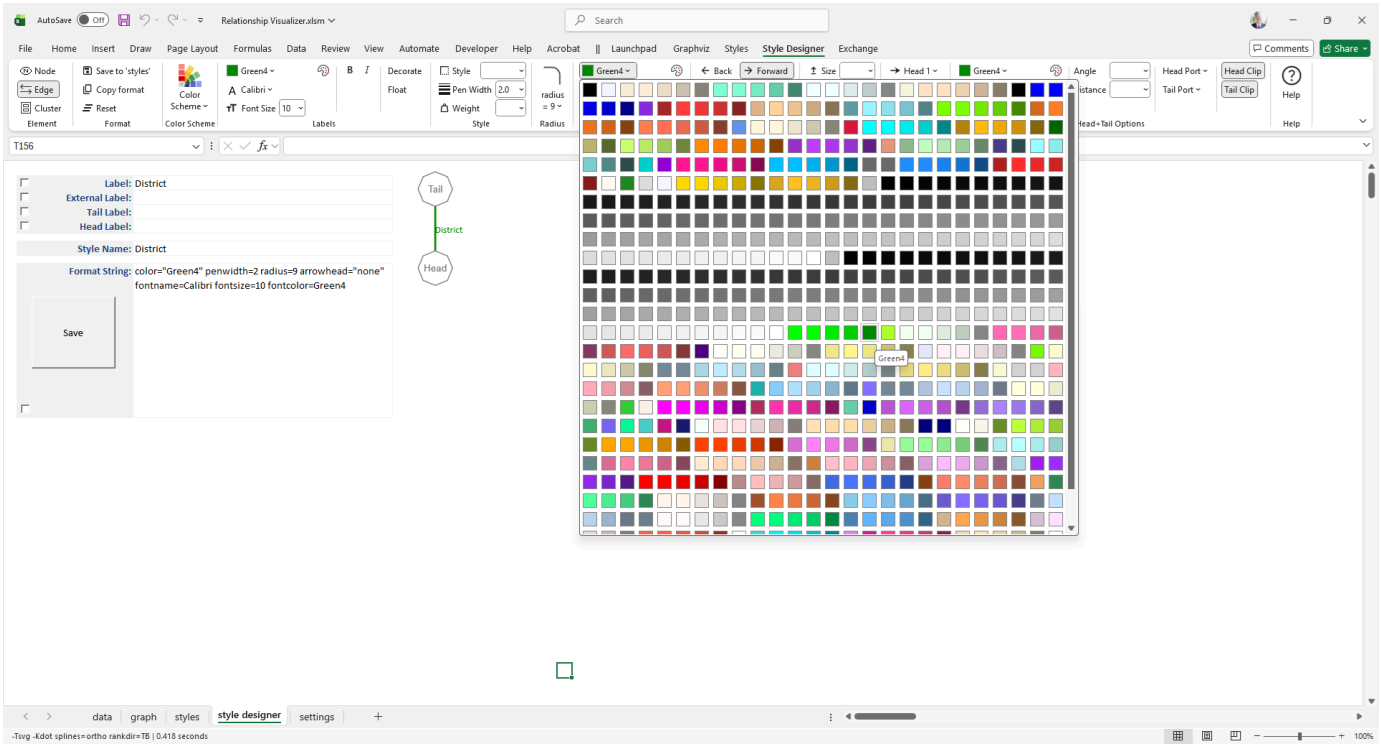


[Learn more...]

Design Styles

The Relationship Visualizer makes it easy to create combinations of Graphviz style attributes which can be applied to nodes, edges, or clusters. It provides a `style designer` worksheet to define the styles, and a `styles` worksheet to provide a gallery of styles which can be selected for a given row on the `data` worksheet (similar to CSS on HTML).

Here the `style designer` is being used to create a style of edge which is dark green (Green4) in color, and does not have an arrow head.

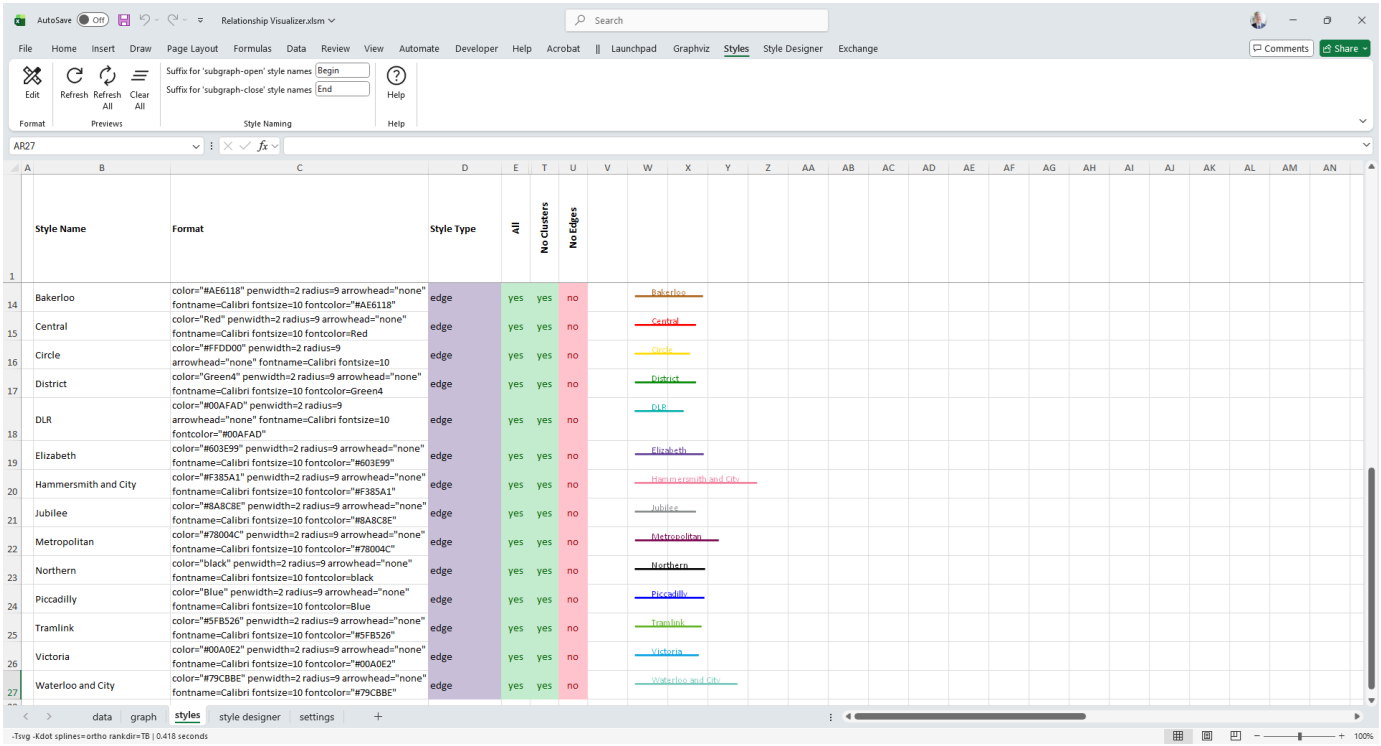


[Learn more...]

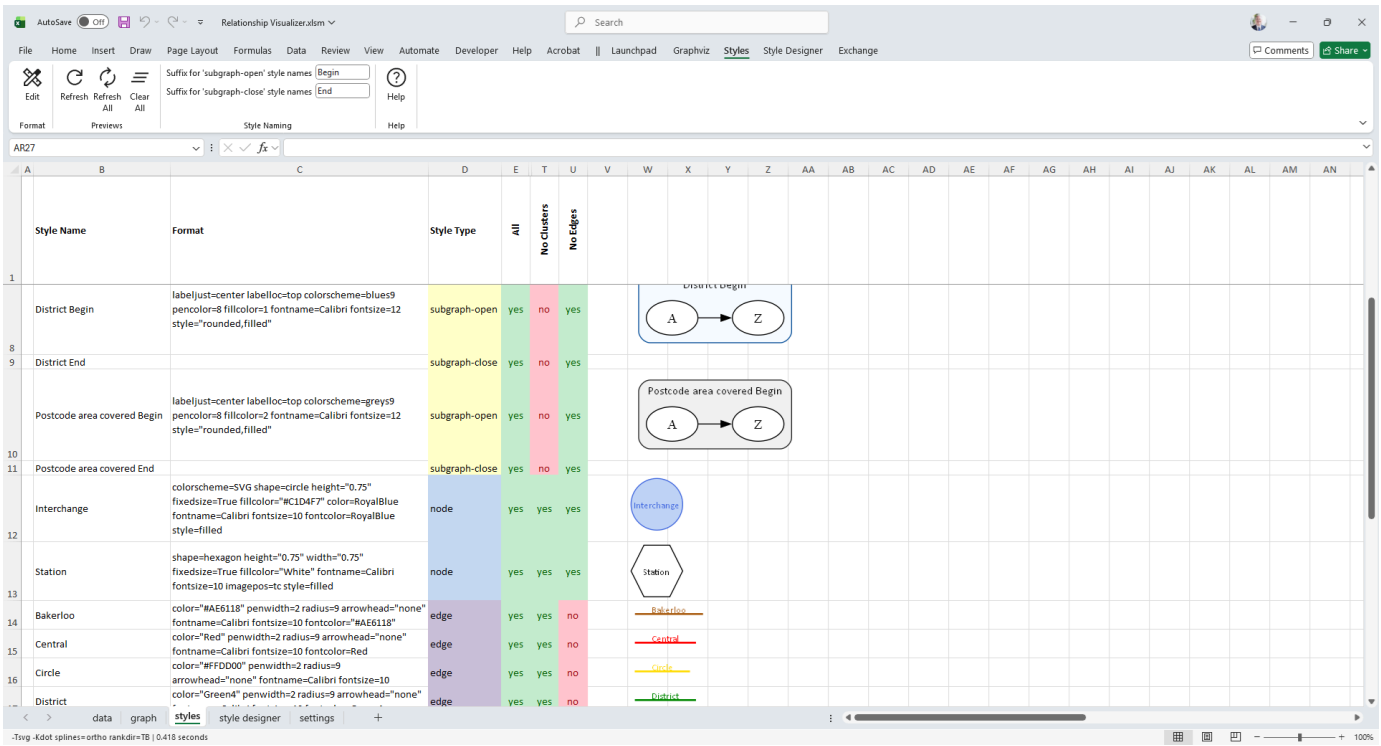
Save Styles

The Green Line on the London Underground is the "District" line. We click the `Add to 'styles'` button. On the `styles` worksheet the format string is saved on a row named **District** along with a preview image showing what the format string looks like.

Notice that Style Names have been created for each of the subway lines, and set to their designated color. We will see them used just ahead.



Cluster and node formats are created in similar fashion. This image shows the other elements which have been defined to depict the London Underground. For example, stations are depicted with a square shape, and interchanges are depicted by a circle.

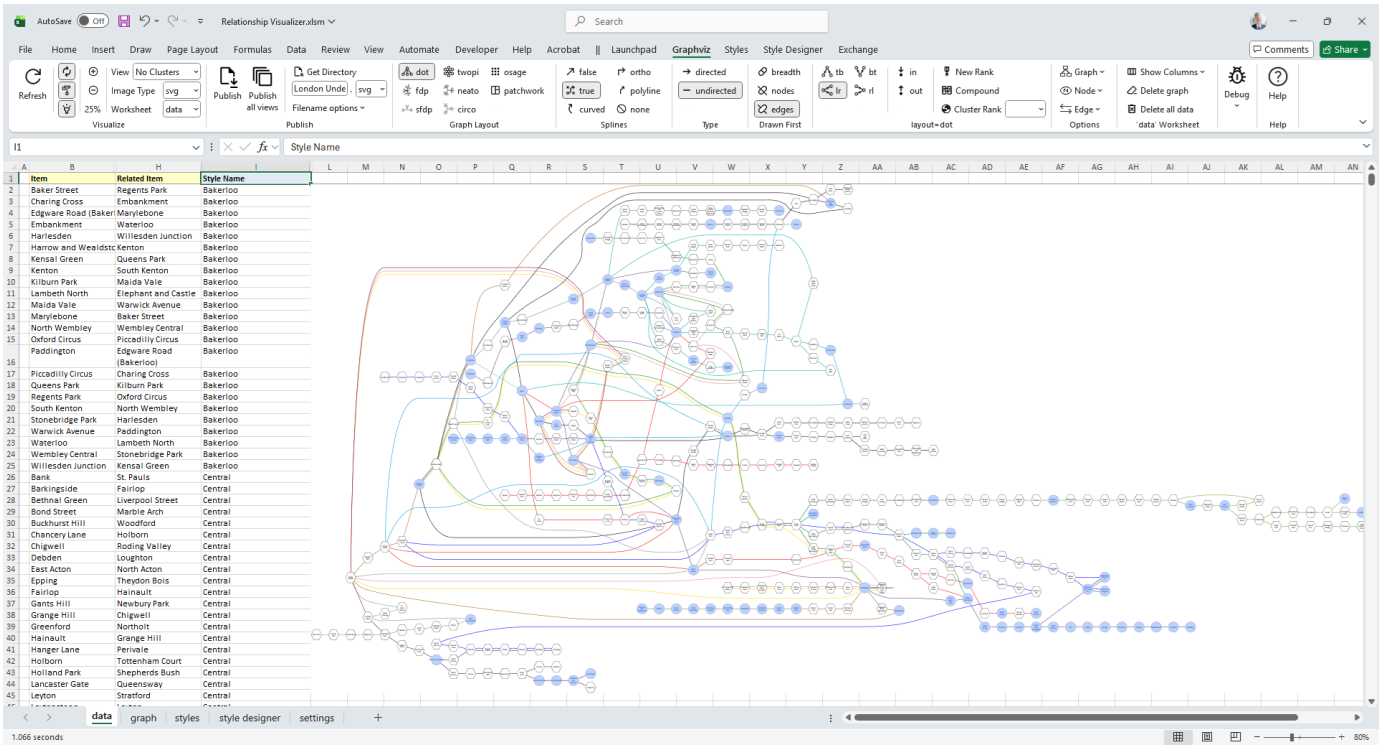


[Learn more...]

Apply Styles

Styles are applied to data rows via a `Style Name` column on the `data` worksheet. Style Names selected from a dropdown list populated with the names on the `styles` worksheet.

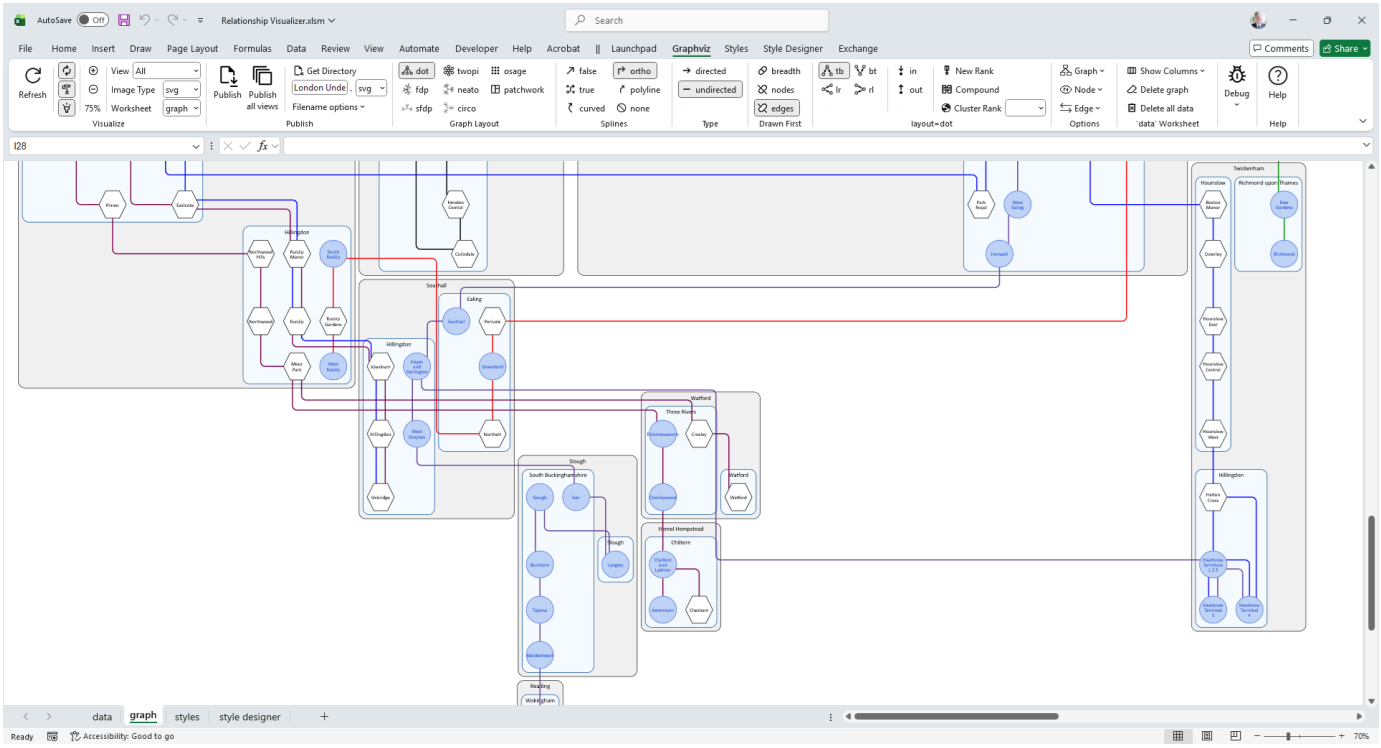
In the image below each subway line is now depicted in its assigned colors, and shapes imply if a stop is a regular station, or an interchange.



Cluster Elements

The Relationship Visualizer lets you easily cluster nodes by adding rows containing `{` and `}` at the start and end of the cluster respectively. Clusters can have assigned styles, and clusters may contain other clusters.

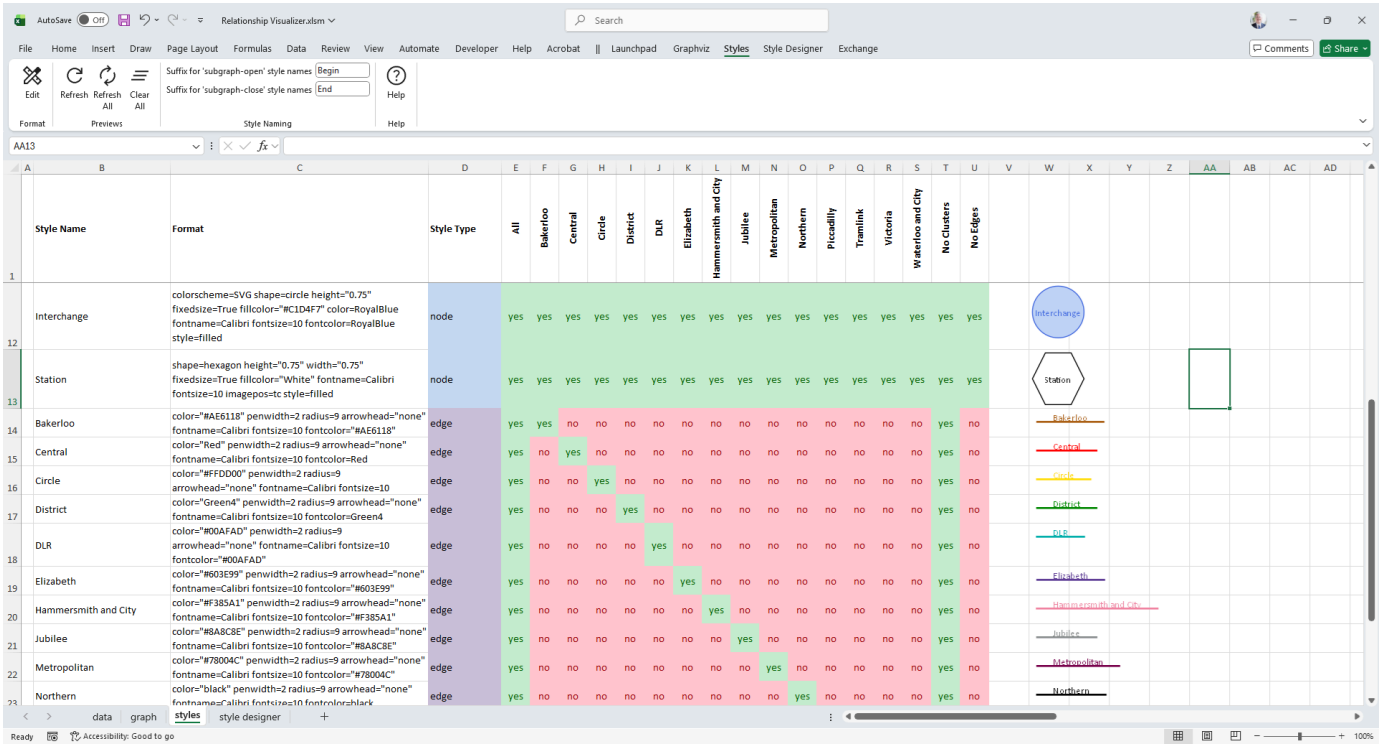
This graph depicts different areas of London, and the postal districts in which the stations are located.



Split Data into Views

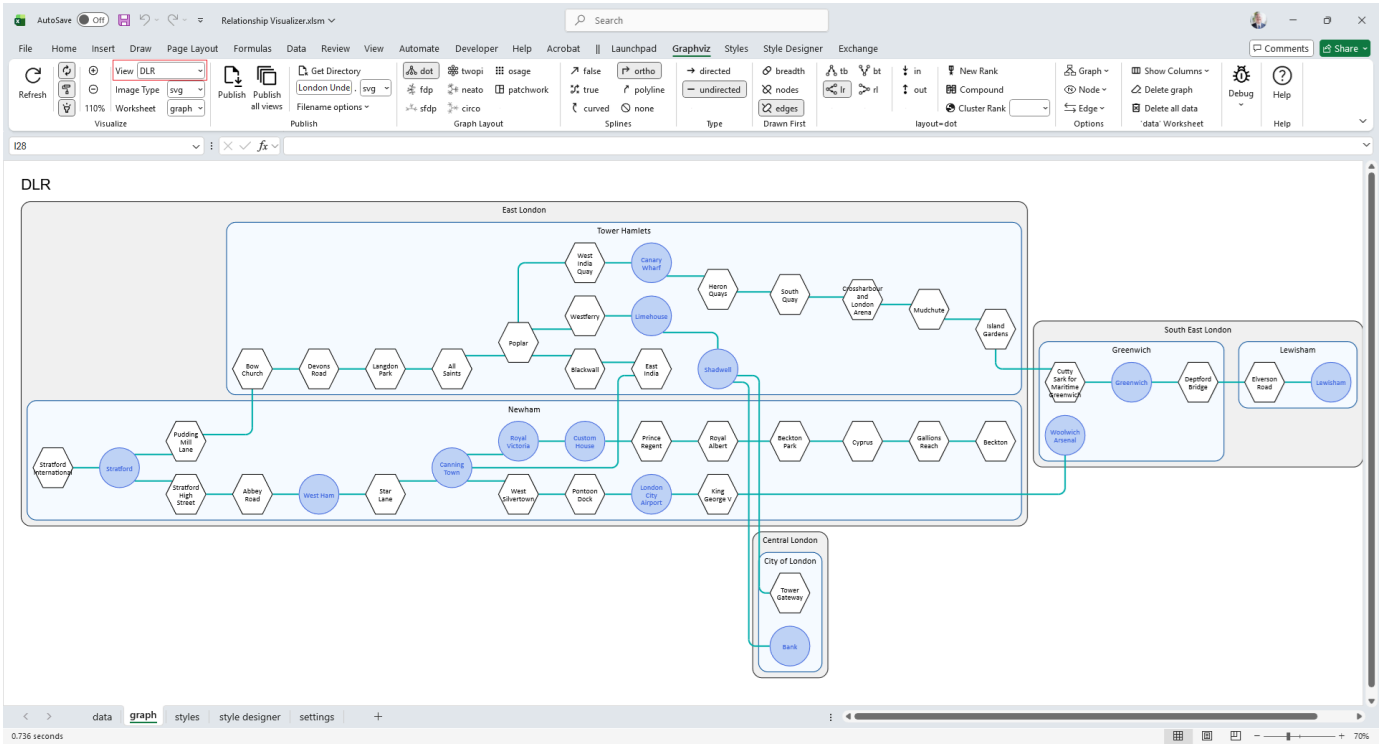
The Relationship Visualizer allows you to have multiple views of the same data by enabling or disabling styles on the `styles` worksheet. It generates graphs that include only the elements of the enabled styles.

In this example the `styles` worksheet modified to have one column per subway line, which only enables the styles for that subway line.



The column names appear in a `View` dropdown list on the `Graphviz` tab. Selecting a view name redraws the graph to include only the styles where a value of yes is present in that column.

In the image below, selecting the "DLR" subway line from the `View` dropdown highlights the areas of London served by the DLR and its stations, while filtering out the rest of the London Underground.



Learn more...

Publish Graphs

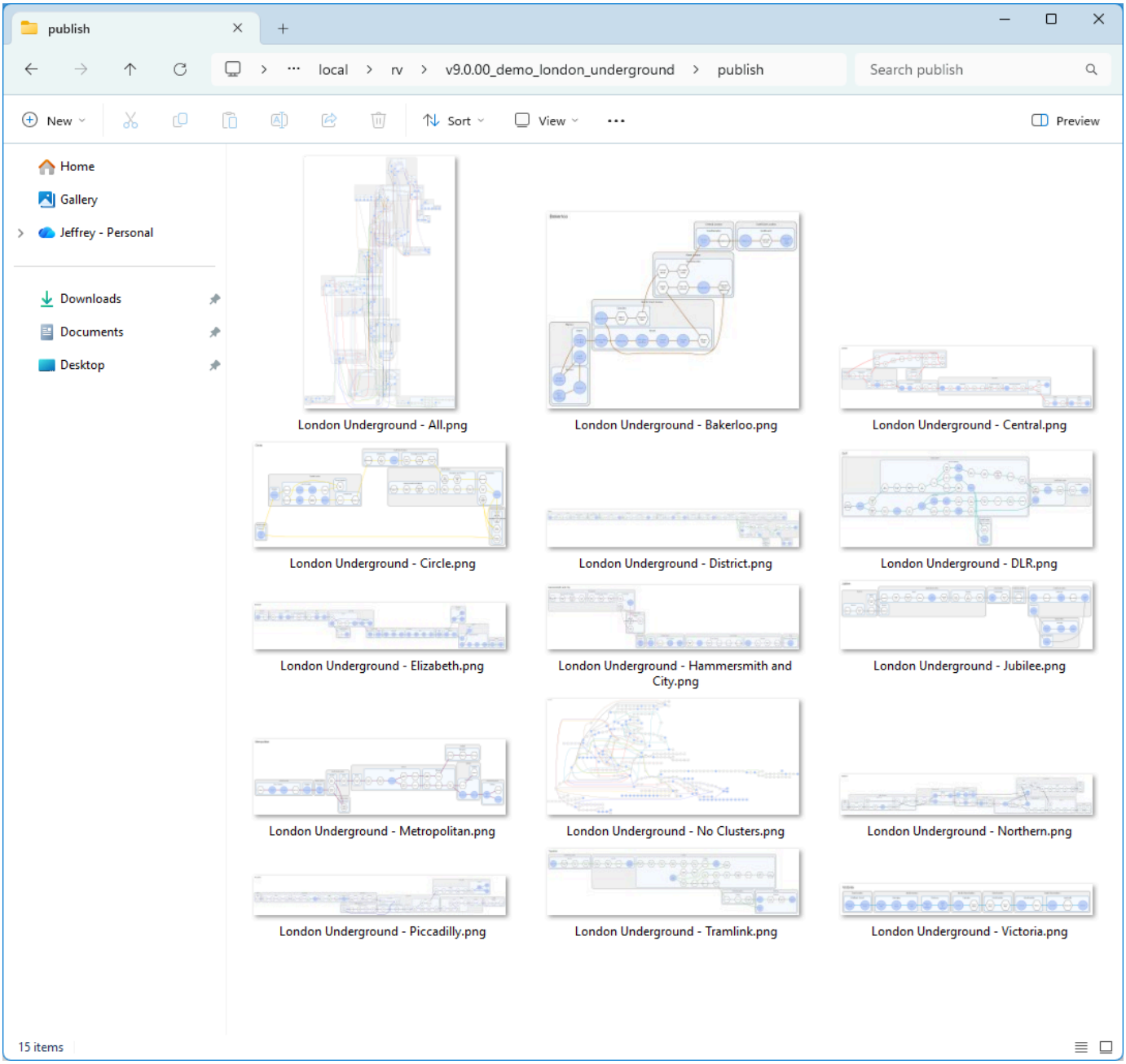
Graph to File

Once you have created a graph to your satisfaction, a push of a button creates the graph as an external file.

You can publish graphs in the following formats: `bmp` , `gif` , `jpg` , `pdf` , `png` , `ps` , `svg` , `tiff` , `json` , `dot`

Publish All Views to File

You can publish a single graph, or use the `Publish all views` button to automatically cycle through all the view names, publishing each view as its own file. In this example, Windows Explorer displays the results after all the various London Underground lines have been published as graphs.



Learn more...

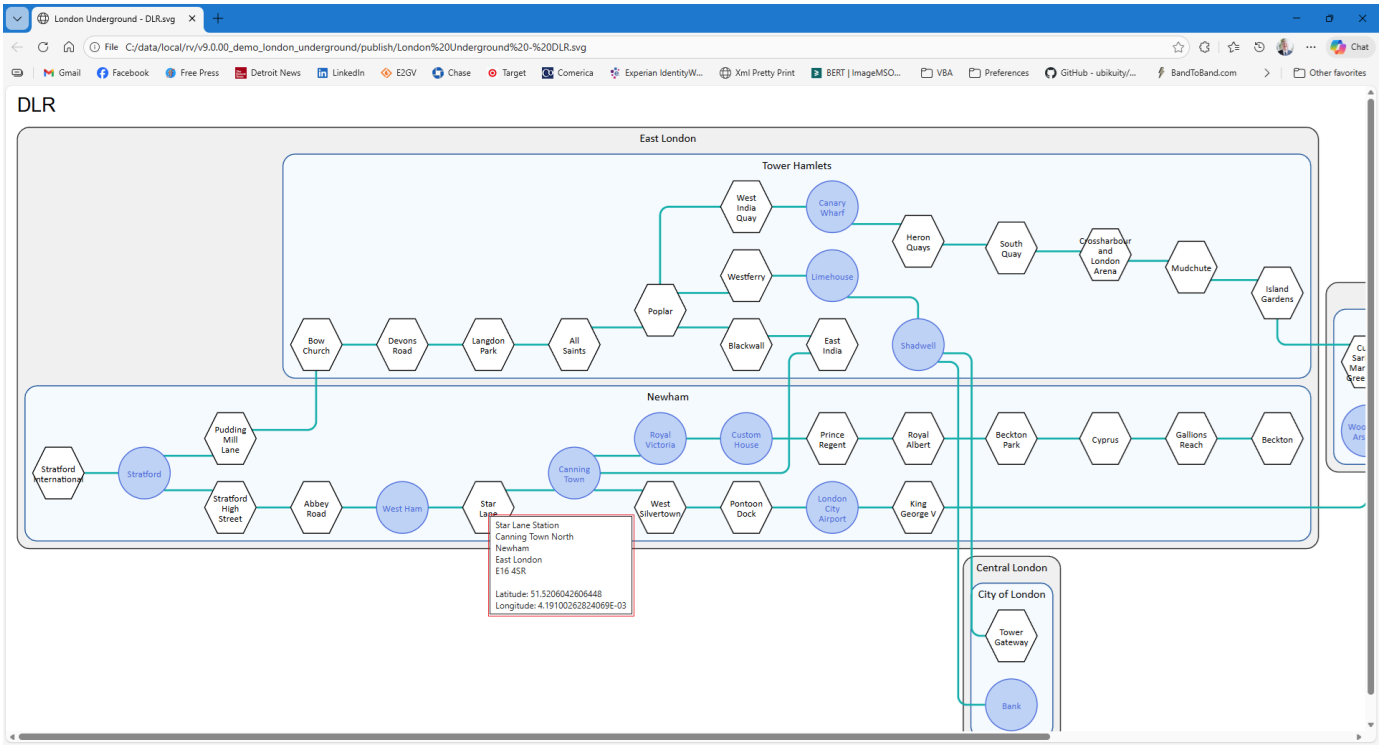
Enhance SVG Output

SVG, or Scalable Vector Graphics, is an XML-based file format used for creating and displaying vector images that are scalable to any size without losing quality.

Add Tooltips

The Relationship Visualizer provides a tooltips column where you can provide information to be added to the graph to be displayed when a cluster, node, or edge has the mouse hover over it.

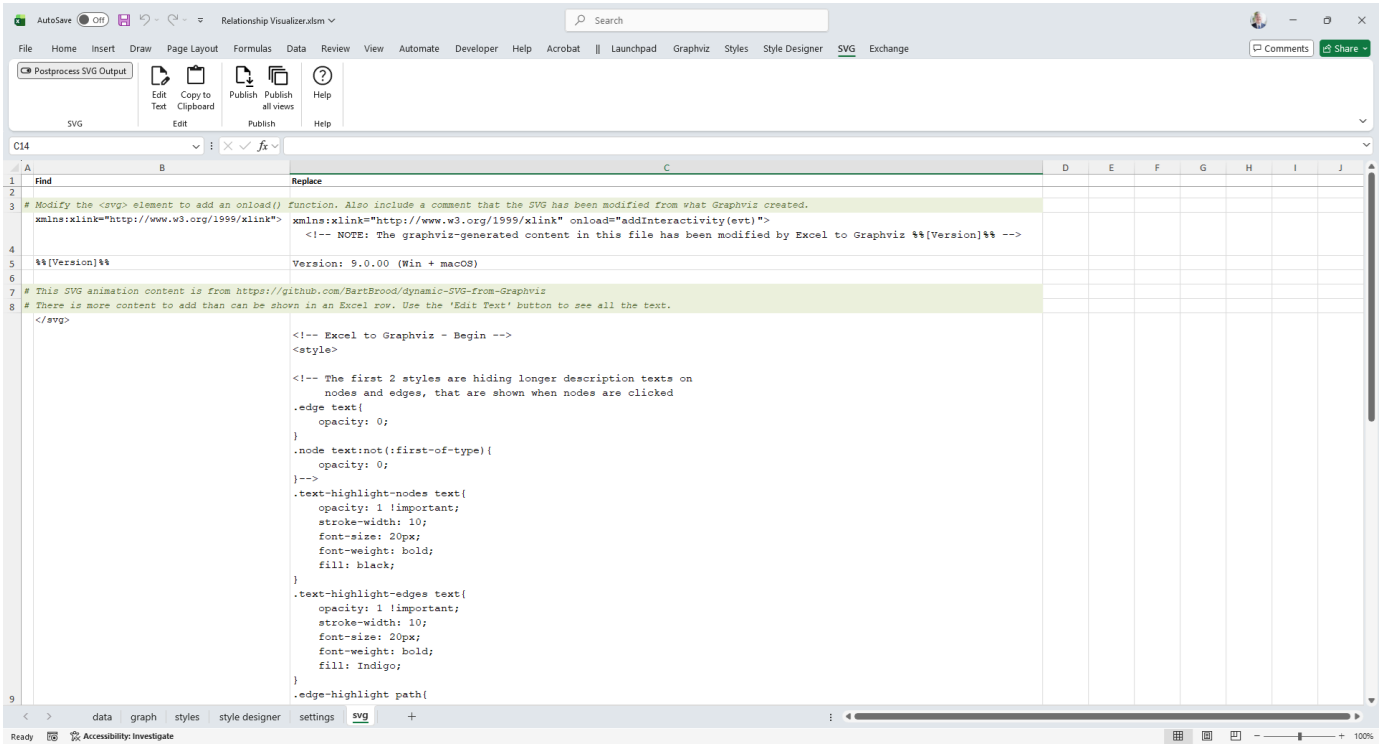
In this example, we see a graph of the DLR Line exported in SVG format. When hovering over the **Star Lane** station, a pop-up displays its village, district, postal code, latitude, and longitude.



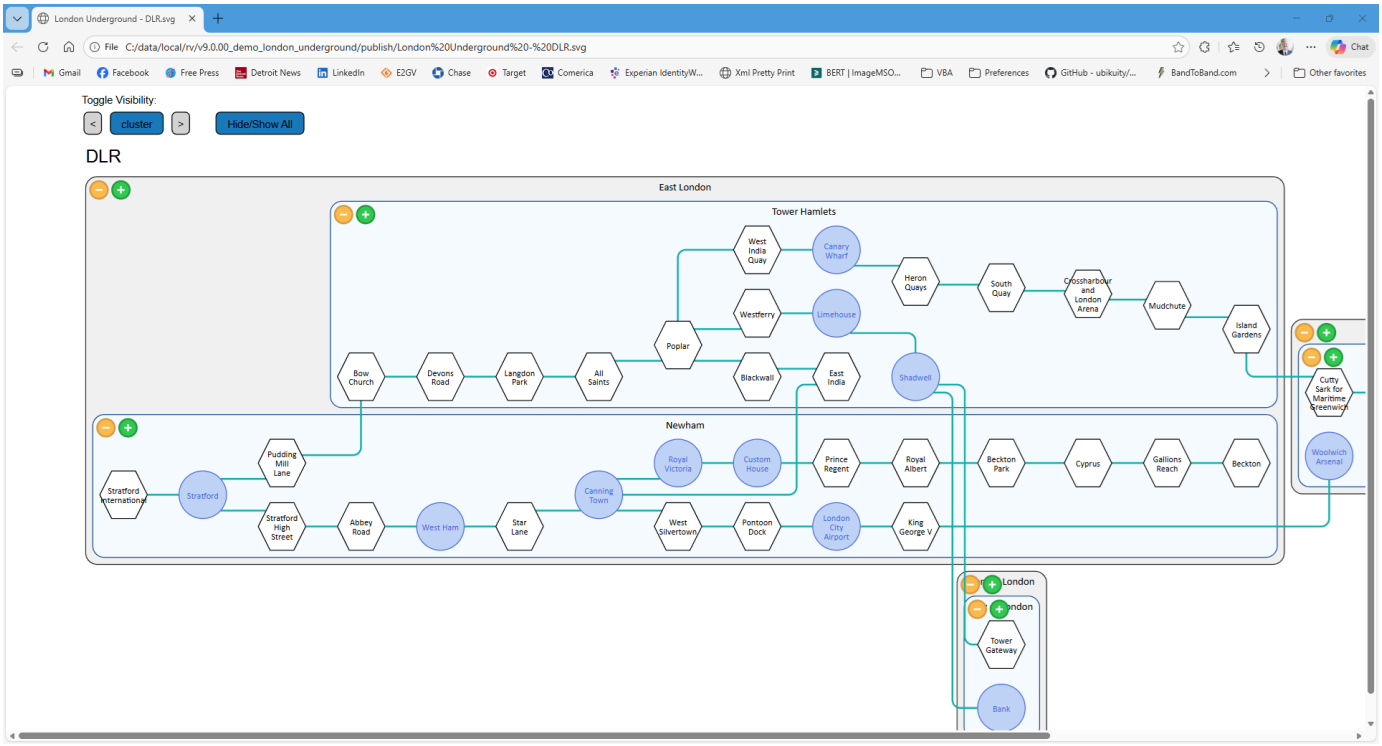
Add Animation

The Relationship Visualizer provides simple post-processing support via a worksheet which can perform find/replace actions against the XML content. Out of the box the Relationship Visualizer can post-process SVG to insert JavaScript routines for add animation capabilities.

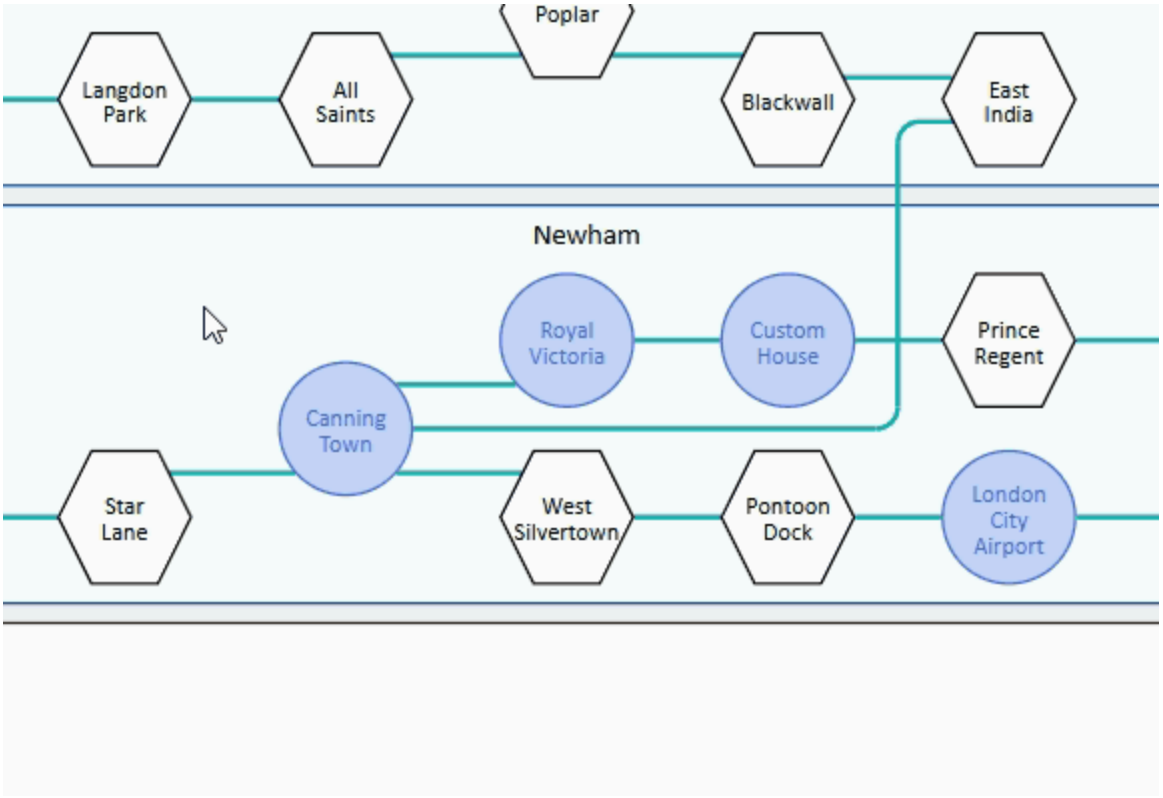
The `svg` worksheet containing the find/replace strings appears as follows:



After post-processing (triggered by graphing to a file), we see that buttons have been added to the SVG file.



And in the animation below, we see how clicking a node triggers an animation to enlarge the node, and animate the edge connections from the node.



Learn more...

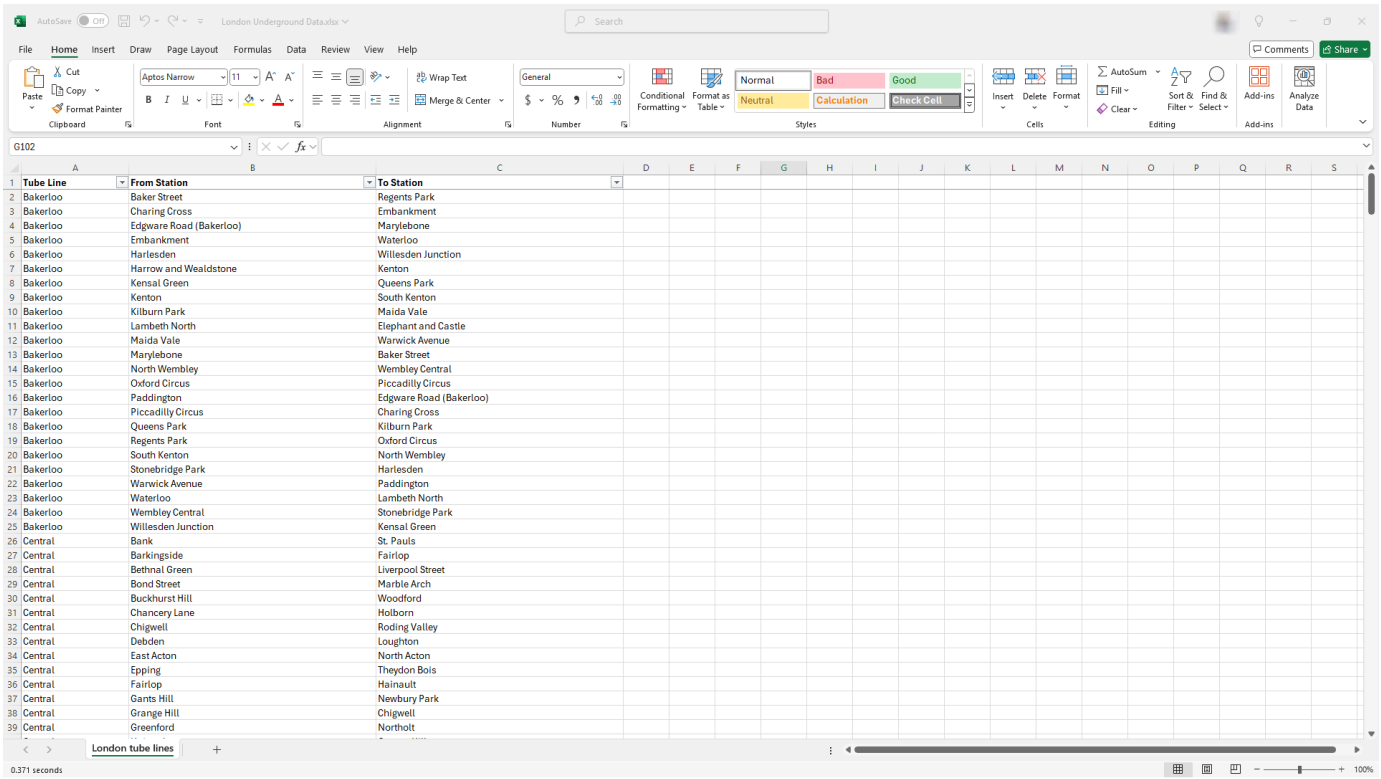
SQL Data Import

Typically, Excel data worksheets are created manually or exported from other applications. With the Relationship Visualizer, creating large graphs from these datasets is straightforward, as it supports importing data from other Excel workbooks and Access databases using SQL.

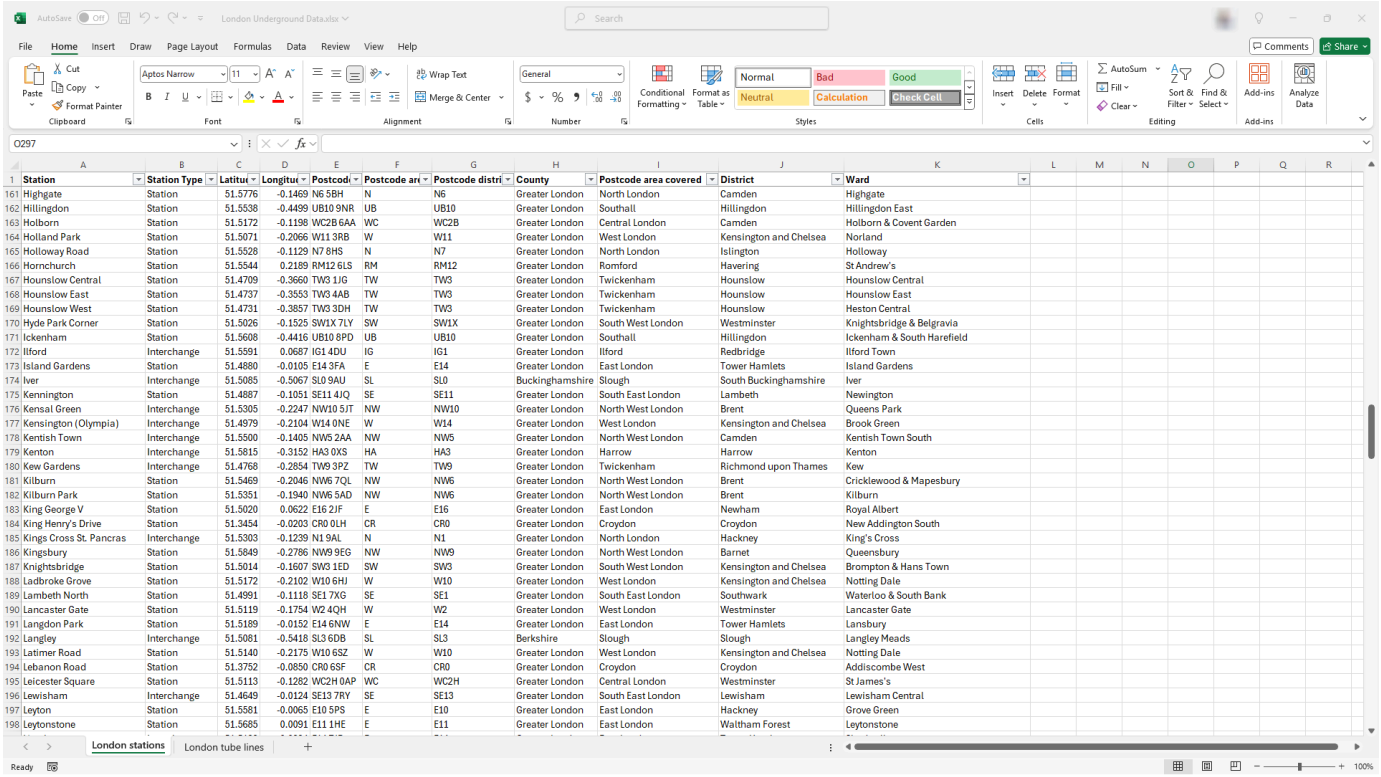
In our London Underground example, we have a `London Underground Data.xlsx` workbook containing two worksheets.

- A `London tube lines` worksheet details which stations connect on each tube line.
- A `London stations` worksheet provides detailed information on each underground station.

They appear as follows:



Tube Line	From Station	To Station
Bakerloo	Baker Street	Regents Park
Bakerloo	Charing Cross	Embankment
Bakerloo	Edgware Road (Bakerloo)	Marylebone
Bakerloo	Embankment	Waterloo
Bakerloo	Harlesden	Willesden Junction
Bakerloo	Harrow and Wealdstone	Kenton
Bakerloo	Kensal Green	Queens Park
Bakerloo	Kenton	South Kenton
Bakerloo	Kilburn Park	Maida Vale
Bakerloo	Lambeth North	Elephant and Castle
Bakerloo	Maida Vale	Warwick Avenue
Bakerloo	Marylebone	Baker Street
Bakerloo	North Wembley	Wembley Central
Bakerloo	Oxford Circus	Piccadilly Circus
Bakerloo	Paddington	Edgware Road (Bakerloo)
Bakerloo	Piccadilly Circus	Charing Cross
Bakerloo	Queens Park	Kilburn Park
Bakerloo	Regents Park	Oxford Circus
Bakerloo	South Kenton	North Wembley
Bakerloo	Stonebridge Park	Harlesden
Bakerloo	Warwick Avenue	Paddington
Bakerloo	Waterloo	Lambeth North
Bakerloo	Wembley Central	Stonebridge Park
Bakerloo	Willesden Junction	Kensal Green
Central	Bank	St. Pauls
Central	Barkingside	Fairlop
Central	Bethnal Green	Liverpool Street
Central	Bond Street	Marble Arch
Central	Buckhurst Hill	Woodford
Central	Chancery Lane	Holborn
Central	Chigwell	Roding Valley
Central	Debden	Loughton
Central	East Acton	North Acton
Central	Epping	Theydon Bois
Central	Fairlop	Hainault
Central	Gants Hill	Newbury Park
Central	Grange Hill	Chigwell
Central	Greenford	Northolt



Station	Station Type	Latitude	Longitude	Postcode	Postcode area	Postcode district	County	Postcode area covered	District	Ward
Highgate	Station	51.5776	-0.1469	N6 5BH	N	N6	Greater London	North London	Camden	Highgate
Hillingdon	Station	51.5538	-0.4499	UB10 9NR	UB	UB10	Greater London	Southall	Hillingdon	Hillingdon East
Holborn	Station	51.5172	-0.1198	WC2B 6AA	WC	WC2B	Greater London	Central London	Camden	Holborn & Covent Garden
Holland Park	Station	51.5071	-0.2066	W11 3RB	W	W11	Greater London	West London	Kensington and Chelsea	Norland
Holloway Road	Station	51.5528	-0.1129	N7 8HS	N	N7	Greater London	North London	Islington	Holloway
Hornchurch	Station	51.5544	0.2189	RM12 6LS	RM	RM12	Greater London	Romford	Havering	St Andrew's
Hounslow Central	Station	51.4709	-0.3660	TW3 1JG	TW	TW3	Greater London	Twickenham	Hounslow	Hounslow Central
Hounslow East	Station	51.4737	-0.3553	TW3 4AB	TW	TW3	Greater London	Twickenham	Hounslow	Hounslow East
Hounslow West	Station	51.4731	-0.3857	TW3 3DH	TW	TW3	Greater London	Twickenham	Hounslow	Heston Central
Hyde Park Corner	Station	51.5026	-0.1525	SW1X 7LY	SW	SW1X	Greater London	South West London	Westminster	Knightsbridge & Belgravia
Ickenham	Station	51.5608	-0.4416	UB10 8PD	UB	UB10	Greater London	Southall	Hillingdon	Ickenham & South Harefield
Ilford	Interchange	51.5591	0.0687	IG1 4DU	IG	IG1	Greater London	Ilford	Redbridge	Ilford Town
Island Gardens	Station	51.4880	-0.0105	E14 3FA	E	E14	Greater London	East London	Tower Hamlets	Island Gardens
Iver	Interchange	51.5085	-0.5067	SL0 9AU	SL	SL0	Buckinghamshire	Slough	South Buckinghamshire	Iver
Kennington	Station	51.4887	-0.1051	SE11 4JQ	SE	SE11	Greater London	South East London	Lambeth	Newington
Kensal Green	Interchange	51.5305	-0.2247	NW10 5JT	NW	NW10	Greater London	North West London	Brent	Queens Park
Kensington (Olympia)	Interchange	51.4979	-0.2104	W14 0NE	W	W14	Greater London	West London	Kensington and Chelsea	Brook Green
Kentish Town	Interchange	51.5500	-0.1405	NW5 2AA	NW	NW5	Greater London	North West London	Camden	Kentish Town South
Kenton	Interchange	51.5815	-0.3152	HA3 0XS	HA	HA3	Greater London	Harrow	Harrow	Kenton
Kew Gardens	Interchange	51.4768	-0.2854	TW9 3PZ	TW	TW9	Greater London	Twickenham	Richmond upon Thames	Kew
Kilburn	Station	51.5469	-0.2046	NW6 7QL	NW	NW6	Greater London	North West London	Brent	Cricklewood & Mapesbury
Kilburn Park	Station	51.5351	-0.1940	NW6 5AD	NW	NW6	Greater London	North West London	Brent	Kilburn
King George V	Station	51.5020	0.0622	E16 2JF	E	E16	Greater London	East London	Newham	Royal Albert
King Henry's Drive	Station	51.3454	-0.0203	CR0 0LH	CR	CR0	Greater London	Croydon	Croydon	New Addington South
Kings Cross St. Pancras	Interchange	51.5303	-0.1239	N1 9AL	N	N1	Greater London	North London	Hackney	King's Cross
Kingsbury	Station	51.5849	-0.2786	NW9 9EG	NW	NW9	Greater London	North West London	Barnet	Queensbury
Knightsbridge	Station	51.5014	-0.1607	SW3 1ED	SW	SW3	Greater London	South West London	Kensington and Chelsea	Brompton & Hans Town
Ladbroke Grove	Station	51.5172	-0.2102	W10 6HJ	W	W10	Greater London	West London	Kensington and Chelsea	Notting Dale
Lambeth North	Station	51.4991	-0.1118	SE1 7XG	SE	SE1	Greater London	South East London	Southwark	Waterloo & South Bank
Lancaster Gate	Station	51.5119	-0.1754	W2 4QH	W	W2	Greater London	West London	Westminster	Lancaster Gate
Langdon Park	Station	51.5189	-0.0152	E14 6NW	E	E14	Greater London	East London	Tower Hamlets	Lansbury
Langley	Interchange	51.5081	-0.5418	SL3 6DB	SL	SL3	Berkshire	Slough	Slough	Langley Meads
Latimer Road	Station	51.5140	-0.2175	W10 6SZ	W	W10	Greater London	West London	Kensington and Chelsea	Notting Dale
Lebanon Road	Station	51.3752	-0.0850	CR0 6SF	CR	CR0	Greater London	Croydon	Croydon	Addiscombe West
Leicester Square	Station	51.5113	-0.1282	WC2H 0AP	WC	WC2H	Greater London	Central London	Westminster	St James's
Lewisham	Interchange	51.4649	-0.0124	SE13 7RY	SE	SE13	Greater London	South East London	Lewisham	Lewisham Central
Leyton	Station	51.5581	-0.0065	E10 5PS	E	E10	Greater London	East London	Hackney	Grove Green
Leytonstone	Station	51.5685	0.0091	E11 1HE	E	E11	Greater London	East London	Waltham Forest	Leytonstone

We write a SQL statement to extract data for **edge** relationships:

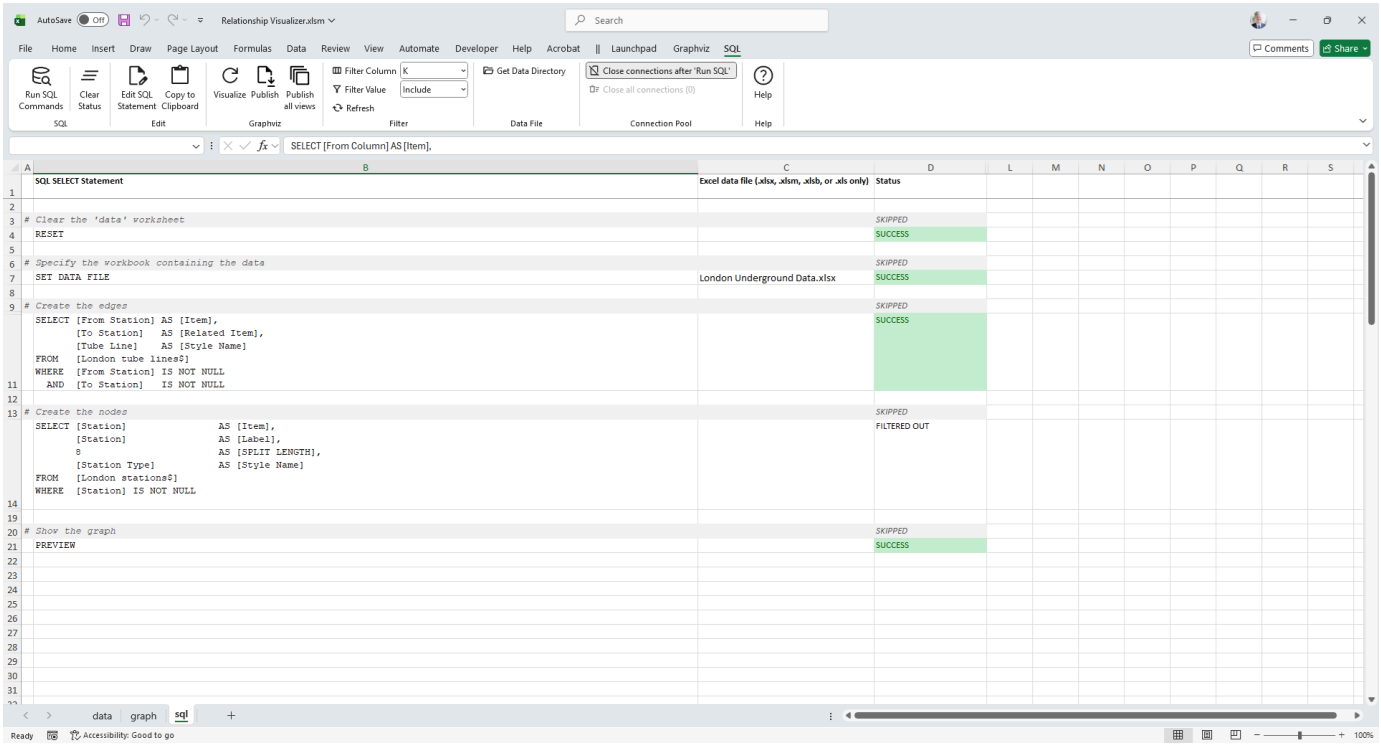
```sql
SELECT [From Station] AS [Item], [To Station] AS [Related Item], [Tube Line] AS [Style
```

```
FROM   [London tube lines$]
```

And we write a SQL statement to extract data for **nodes**.

```sql
SELECT [Station] AS [Item], [Station] AS [Label], [Station Type] AS [Style Name]
FROM   [London stations$]
```

The `sql` worksheet in the Relationship Visualizer appears as:



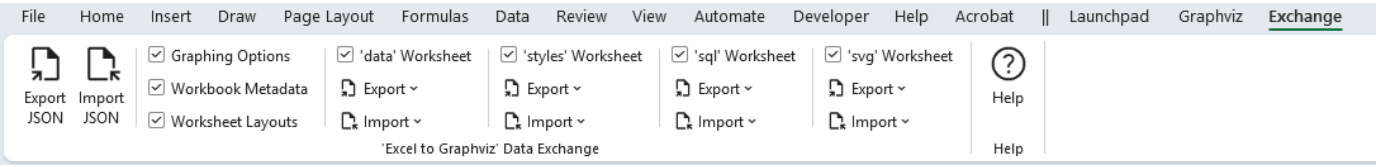
Pressing a button executes the SQL statements, which import the data and map it to the appropriate columns for visualization. It really is that simple!

Learn more...

JSON Export/Import

The Relationship Visualizer provides capabilities to generate a text-based representation of the data, styles, and settings in a JSON format. The features supporting the export and

import of spreadsheet data are found in the `Exchange` workbook tab.



Learn more...

Learn the `dot` Language

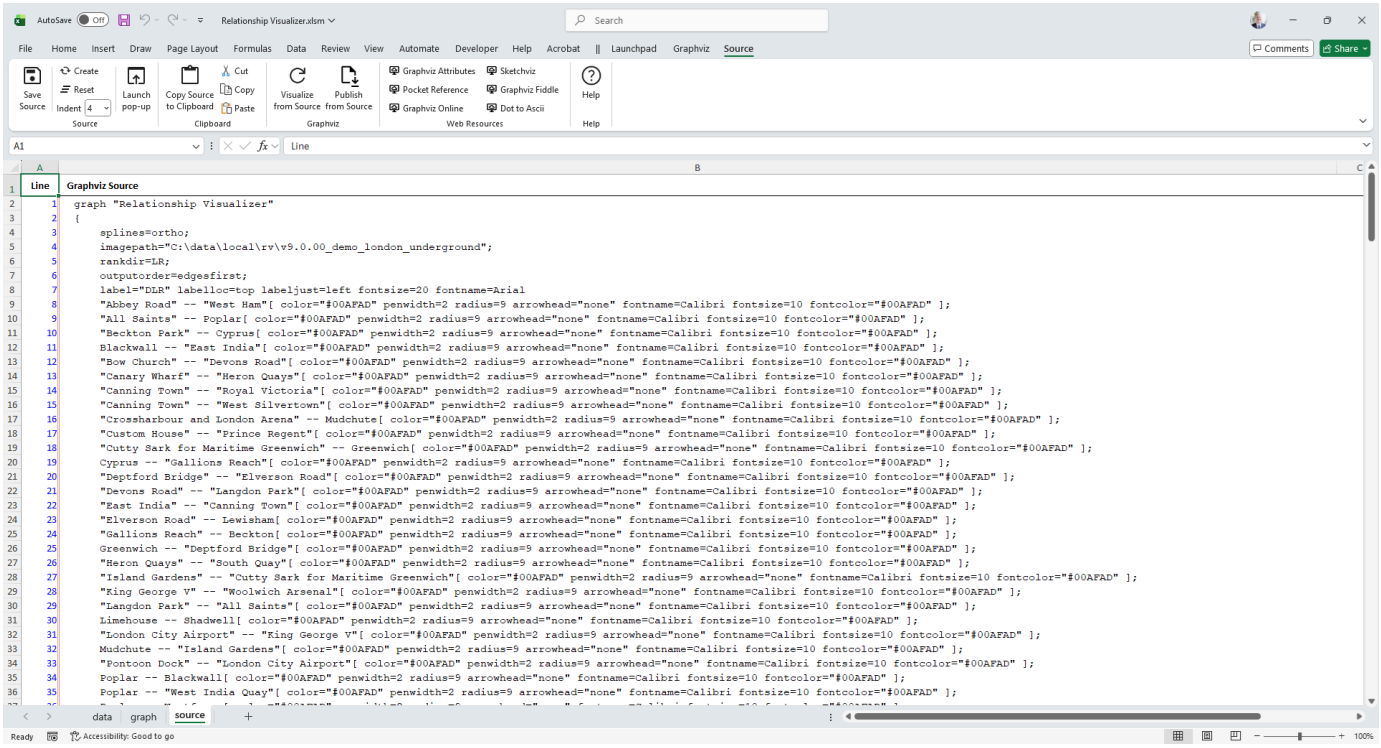
The Relationship Visualizer includes worksheets designed to help you use Graphviz effectively.

View/Save `dot` Source Code

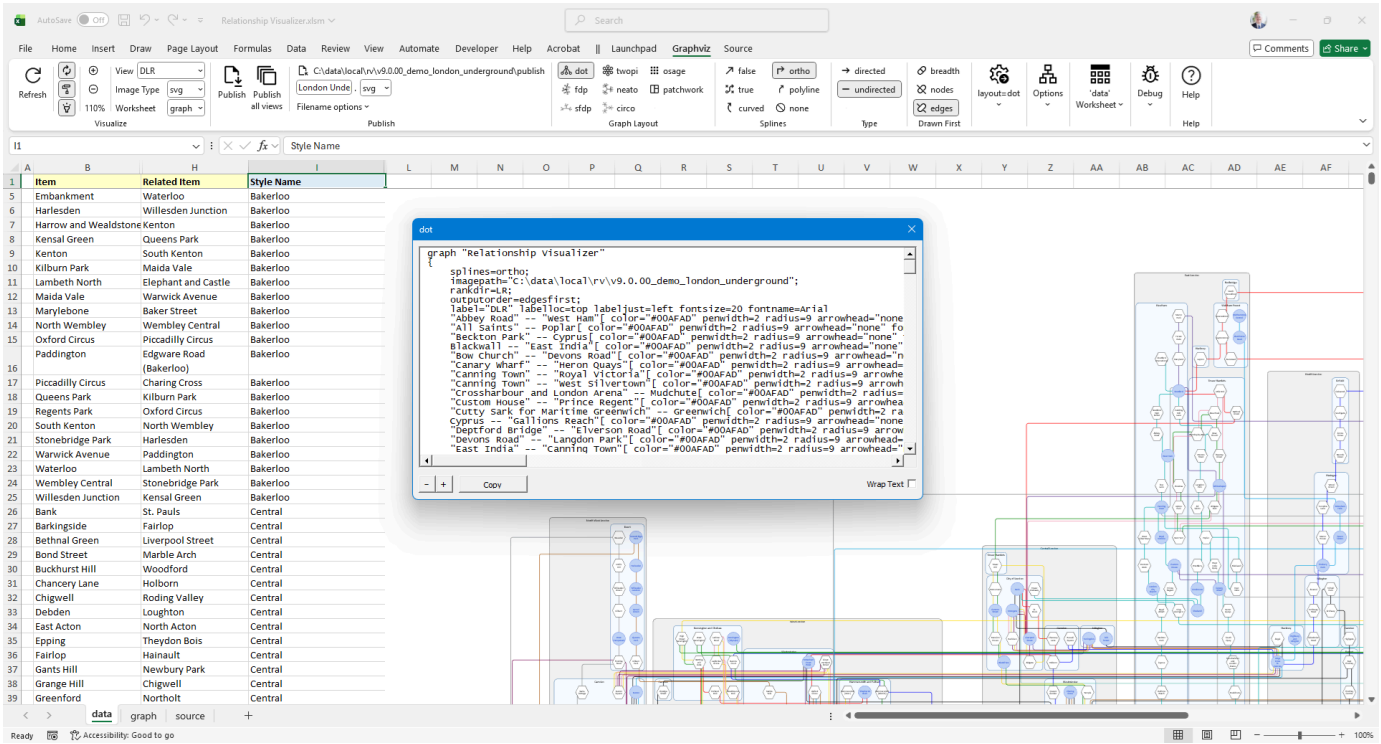
The Relationship Visualizer includes a `source` worksheet that displays the dot source code generated from the data on the `data` worksheet.

You can save this source code to a file or copy it to the clipboard for use in an external Graphviz tool. Links to multiple external Graphviz websites are also provided.

An example appears as follows:



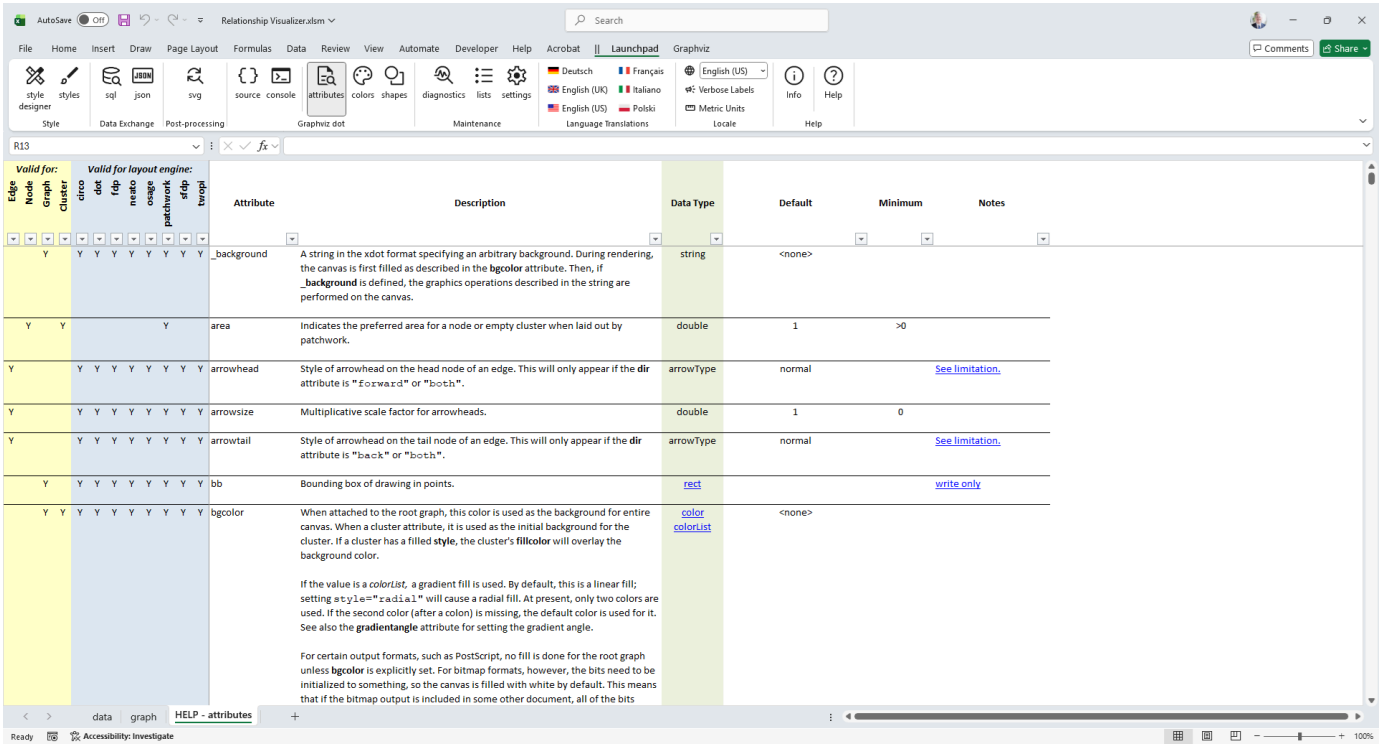
A pop-up source window can be launched, enabling you to view the source while simultaneously examining your data or graph. Below is an example of the source displayed in the pop-up window.



[Learn more...]

Learn `dot` Attributes

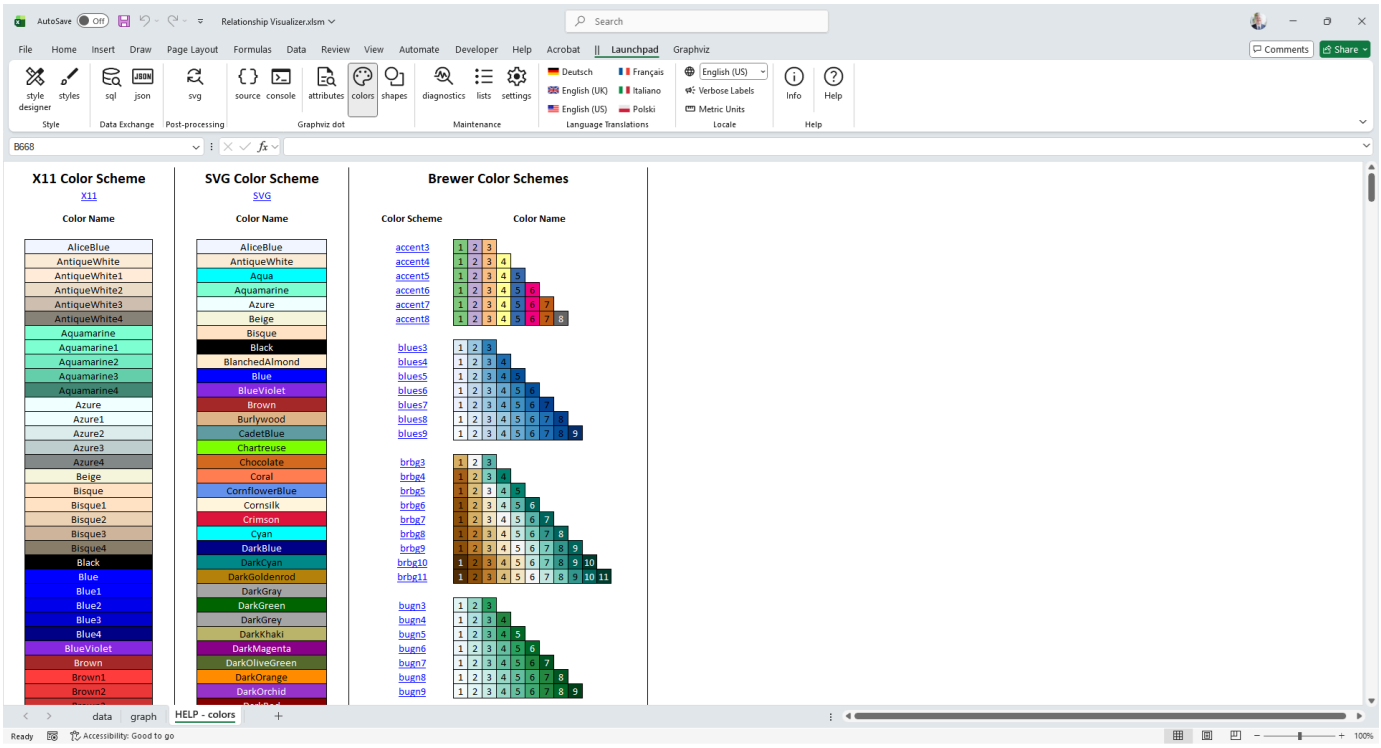
The `HELP - attributes` worksheet provides a list of the `dot` attributes along with columns which can be filtered by layout (dot, fdp, etc.), and attribute type (node, edge, graph).



Valid for: Edge	Node	Graph	Cluster	Valid for layout engine: circo	dot	fdp	neato	osage	patchwork	sfdp	twopi	Attribute	Description	Data Type	Default	Minimum	Notes
	Y			Y	Y	Y	Y	Y	Y	Y	Y	_background	A string in the xdot format specifying an arbitrary background. During rendering, the canvas is first filled as described in the **bgcolor** attribute. Then, if **_background** is defined, the graphics operations described in the string are performed on the canvas.	string	<none>		
Y	Y							Y				area	Indicates the preferred area for a node or empty cluster when laid out by patchwork.	double	1	>0	
Y				Y	Y	Y	Y	Y	Y	Y	Y	arrowhead	Style of arrowhead on the head node of an edge. This will only appear if the **dir** attribute is `"forward"` or `"both"`.	arrowType	normal		See limitation.
Y				Y	Y	Y	Y	Y	Y	Y	Y	arrowsize	Multiplicative scale factor for arrowheads.	double	1	0	
Y				Y	Y	Y	Y	Y	Y	Y	Y	arrowtail	Style of arrowhead on the tail node of an edge. This will only appear if the **dir** attribute is `"back"` or `"both"`.	arrowType	normal		See limitation.
	Y			Y	Y	Y	Y	Y	Y	Y	Y	bb	Bounding box of drawing in points.	rect			write only
	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	bgcolor	When attached to the root graph, this color is used as the background for entire canvas. When a cluster attribute, it is used as the initial background for the cluster. If a cluster has a filled **style**, the cluster's **fillcolor** will overlay the background color. If the value is a *colorList*, a gradient fill is used. By default, this is a linear fill; setting `style="radial"` will cause a radial fill. At present, only two colors are used. If the second color (after a colon) is missing, the default color is used for it. See also the **gradientangle** attribute for setting the gradient angle. For certain output formats, such as PostScript, no fill is done for the root graph unless **bgcolor** is explicitly set. For bitmap formats, however, the bits need to be initialized to something, so the canvas is filled with white by default. This means that if the bitmap output is included in some other document, all of the bits	color colorList	<none>		

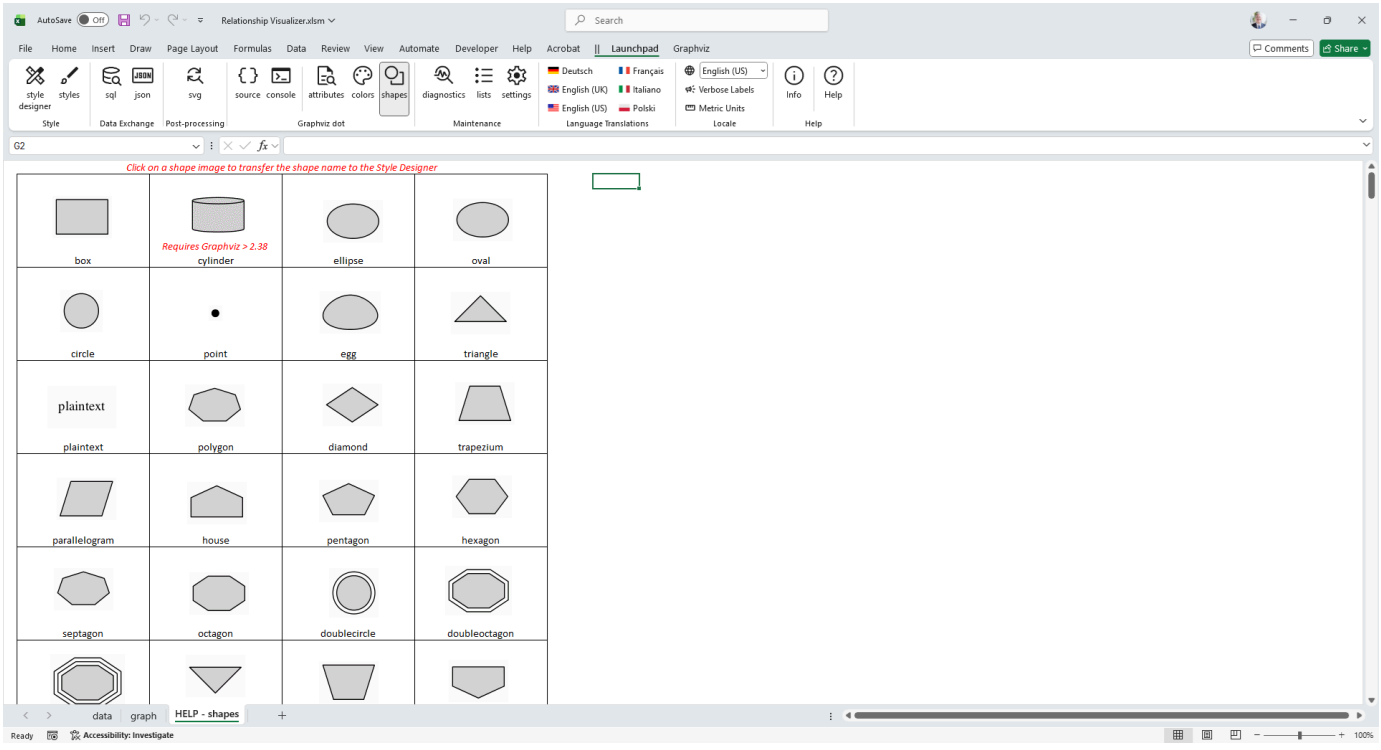
Learn Colors

The `HELP - colors` worksheet provides a visual reference of all the color schemes and colors which Graphviz supports.



Learn Shapes

The `HELP - shapes` worksheet provides a visual reference of all the node shapes which Graphviz supports.

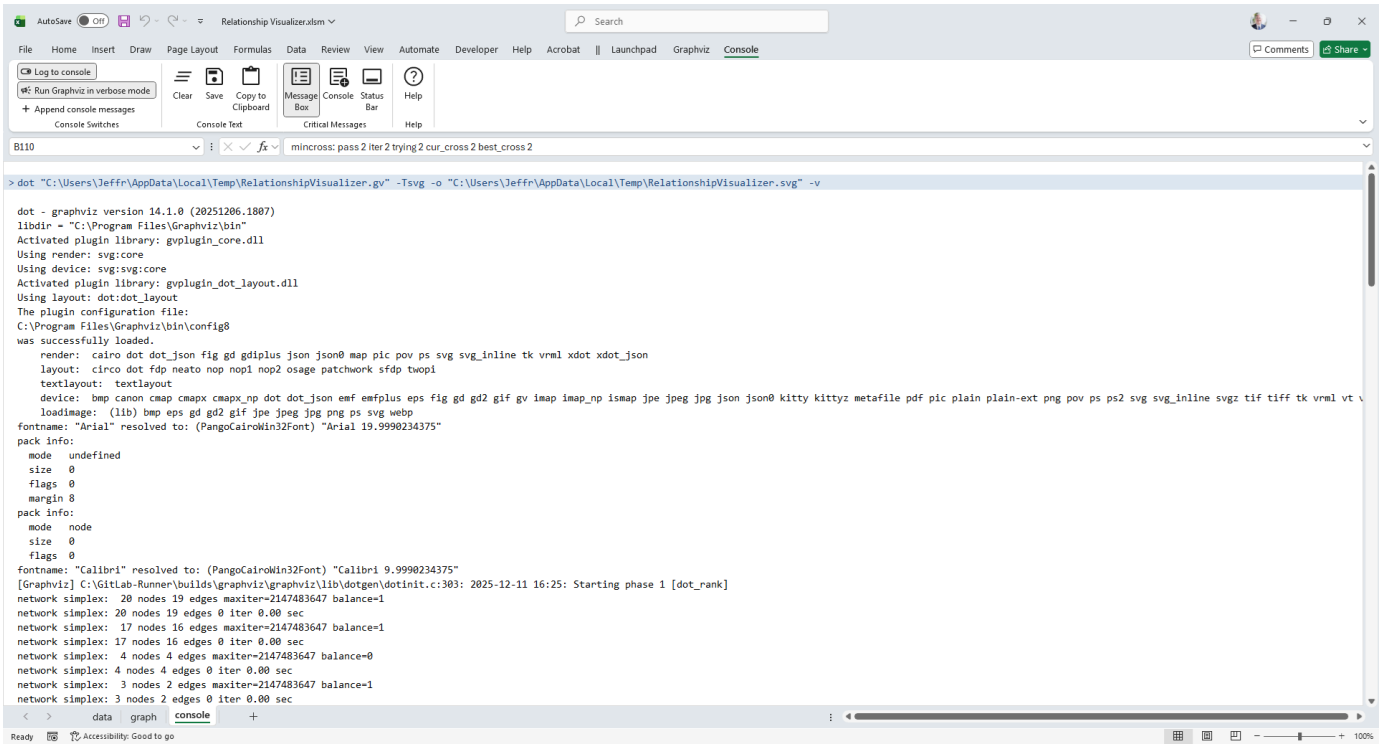


Diagnose Problems

View `dot` Command-line Messages

`dot` is the Graphviz command-line program that the Relationship Visualizer utilizes to generate graphs. A `console` worksheet is available, which can be activated to display the output of the `dot` command.

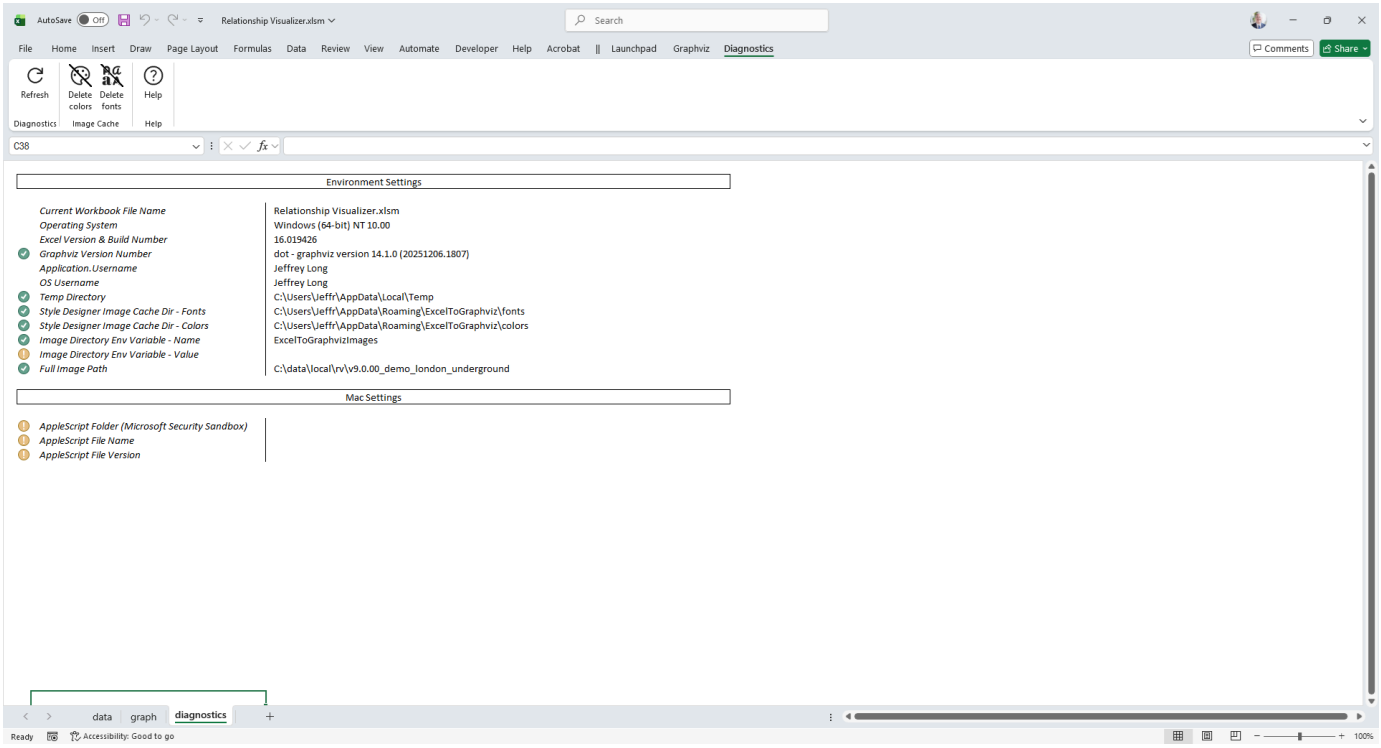
In the example below, `dot` has been executed in `verbose` mode. The `console` worksheet displays `dot`'s diagnostic messages. Notably, the node warnings indicate that two nodes have very long labels exceeding the fixed width of the shape used to depict stations.



Learn more...

View Diagnostic Information

The Relationship Visualizer includes a diagnostics worksheet that provides essential information on software versions and directory paths, useful for troubleshooting and addressing questions.



[Learn more...]

Internationalization

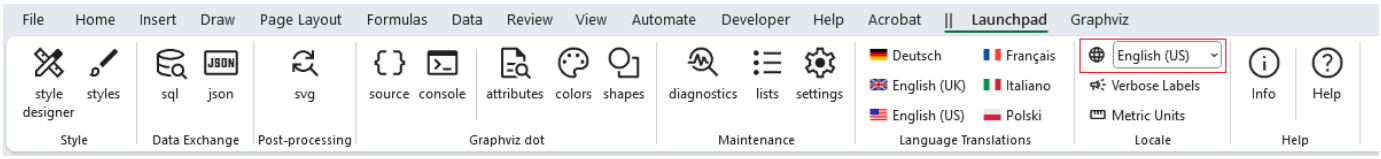
The Relationship Visualizer supports multiple languages, and measurement units.

Set Your Language

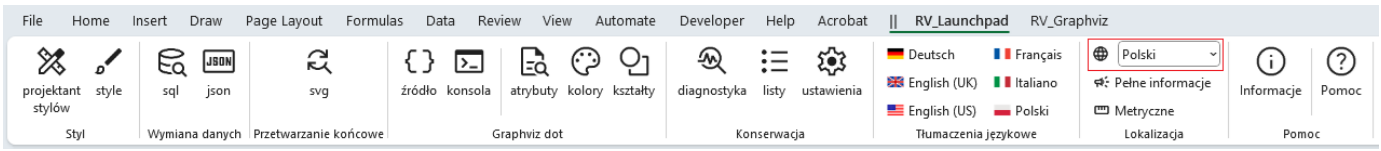
The Relationship Visualizer provides the following language translations:

- English (UK)
- English (US)
- French
- German
- Italian
- Polish

Simply pick your desired language from the `Launchpad` ribbon tab.

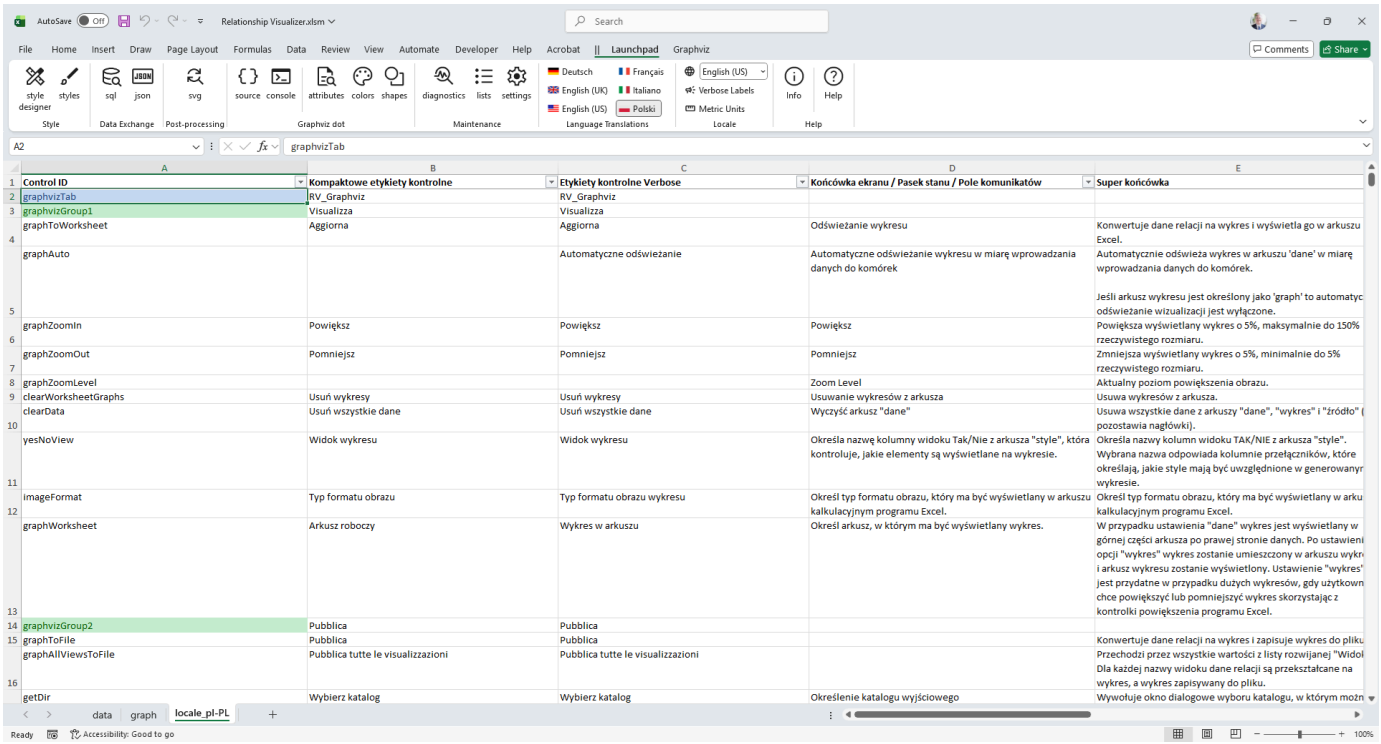


For example, changing the language to Polski (Polish) makes the `Launchpad` ribbon tab appears as:



Edit Language Translations

Translations were performed by machine and may contain minor errors. Fortunately, all language translations are contained in worksheets. From the `Launchpad`, you can unhide the appropriate worksheet and edit the translation as needed.



[Learn more...]

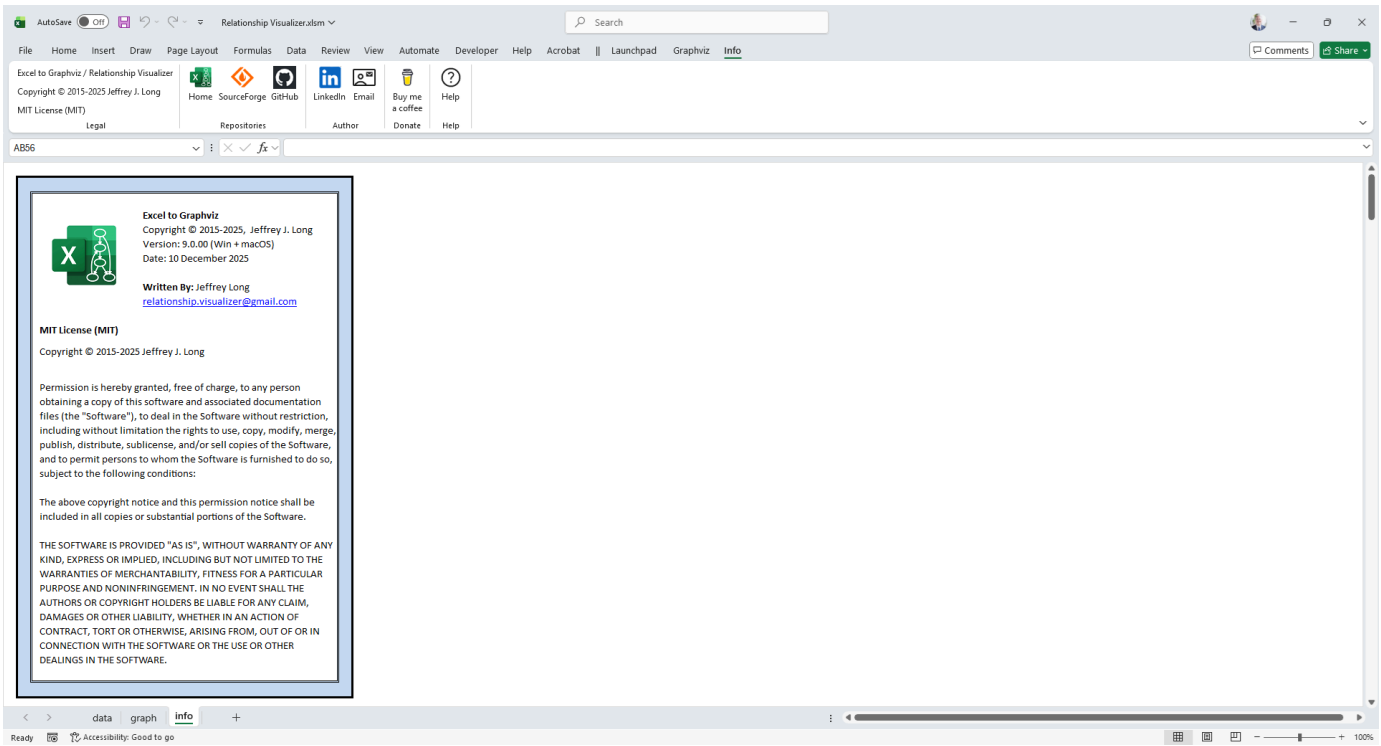
Set Measurement Units

Graphviz uses inches as its unit of measurement for shapes. The `Launchpad` will let you choose Metric units for `style designer` dropdown lists. When `Metric Units` are chosen, measurements in dropdown lists are presented in centimeters, and converted to inches when passed to `dot'.

Information and Acknowledgements

`info` Worksheet

The Relationship Visualizer includes an `info` worksheet that provides details about the spreadsheet, the internal technology incorporated into the tool, and the associated open-source licenses and acknowledgements needed to reuse the open-source components in a license-compliant manner.



[Learn more...]

About...

Copyright, Author, Repositories, and License information are available on this website here.

Thank You

Special thanks to key contributors are acknowledged on this website here as well as in the `info` worksheet.

Third-Party Notices

Open-source licenses and acknowledgements required to reuse the open-source components in a license-compliant manner are available on this website here as well as in the `info` worksheet.

Download

The **Relationship Visualizer** is distributed in a zip file, hosted on SourceForge ↗. SourceForge is utilized for its reliable virus scanning and the availability of checksums, ensuring the downloaded files are secure and free from tampering.

The `RelationshipVisualizer.zip` file contains the main spreadsheet, license information, and several sample workbooks to help you get started. To download it, simply click the large green `Download Now` button below.

Graphviz Required

Relationship Visualizer requires **Graphviz** to be installed on your system. Because download methods and installation differ between Windows and macOS, please refer to the installation instructions for details.

Documentation

If you prefer offline documentation, you can also download a full PDF version of this website. It provides all core pages, examples, and reference material in a single file.

Download Documentation PDF ↗

Installation Instructions

Relationship Visualizer is a single Excel workbook that runs on both Microsoft Windows and Apple macOS[1].

Because each operating system requires a different setup process, use the cards below to jump directly to the instructions for your environment.

Microsoft Windows — *Windows installation instructions*

Apple macOS — *macOS installation instructions*

[1] SQL and Clipboard features are not available on Apple macOS.

Microsoft Windows Installation Instructions

Brief Instructions

Steps to Install **Excel to Graphviz** on Microsoft Windows

1. Download and Install *Graphviz*
   - Choose and run either the 32-bit or 64-bit Graphviz EXE Installer ↗.
   - Ensure the Graphviz `bin` directory is on the `PATH`
2. Open Command Prompt using the `Run as Administrator` option.
   - *Register the Plugins:* Run the command `dot -c` to register Graphviz plugins.
   - *Confirm the Installation:* Run the command `dot -V` to verify the Graphviz version.
3. Download the Relationship Visualizer assets
   - Obtain `RelationshipVisualizer.zip` from SourceForge ↗.
   - *Optional:* Validate SHA1 and/or MD5 checksums available here ↗ against the file downloaded.
4. Extract the files from the Zip file
   - Extract all files to a local directory.
5. Unblock the spreadsheet file
   - Right-click (or `Alt+Enter`) on `Relationship Visualizer.xlsm` and select `Properties`.
   - Check `Unblock` at the bottom of the Properties dialog, then click OK.
6. Open Microsoft Excel
   - Enable Macros in Microsoft Excel's `Trust Center` options.

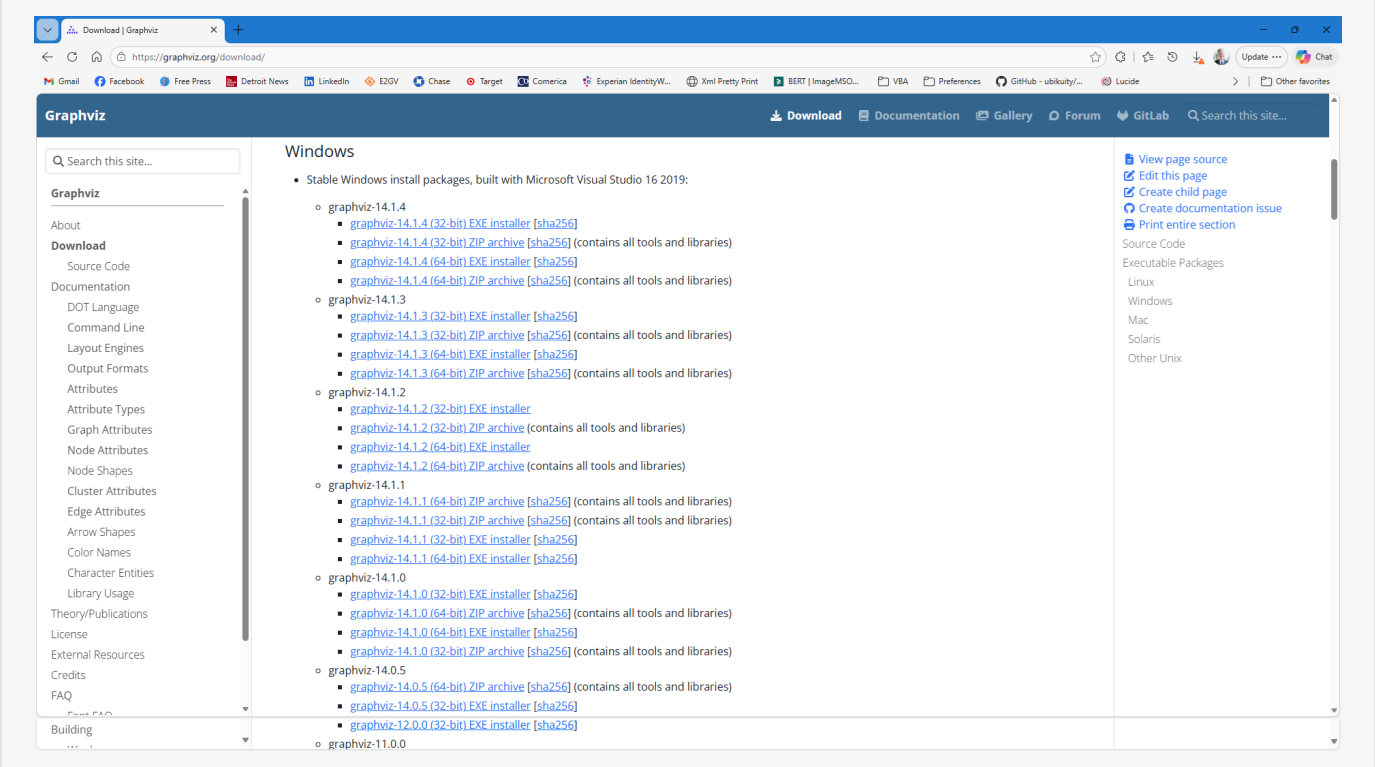
- Open the `Relationship Visualizer.xlsm` workbook, and grant permissions if prompted.
- Save the file as a workbook template

Detailed Instructions

Download and Install Graphviz

Step 1. Download Graphviz Media

Download the installation media from the Windows section of the Graphviz download page at
https://graphviz.org/download/ ↗ which appears as:



You will be presented with two links: Win32 for the 32-bit installer and x64 for the 64-bit installer. Choose the link that corresponds with the Windows architecture of your machine.

Download the installer file. At the time of this writing (25-March-2026) the choices are:

- graphviz-14.1.4 (64-bit) EXE installer ↗
- graphviz-14.1.4 (32-bit) EXE installer ↗

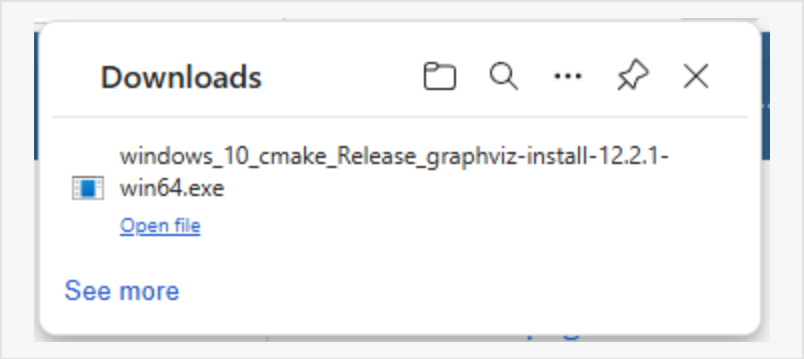
Download blocked?

If you are using Microsoft Edge, ocassionally the download will be blocked. This is usually due to a new release of Graphviz being published and enough downloads have not occurred to tell Microsoft the download is safe.

Follow the steps on this page to learn how to override the block and download the file.

Step 2 - Launch the installer file.

Click on `Open file` to run the installer.



A User Access Control warning will now take over the screen and ask

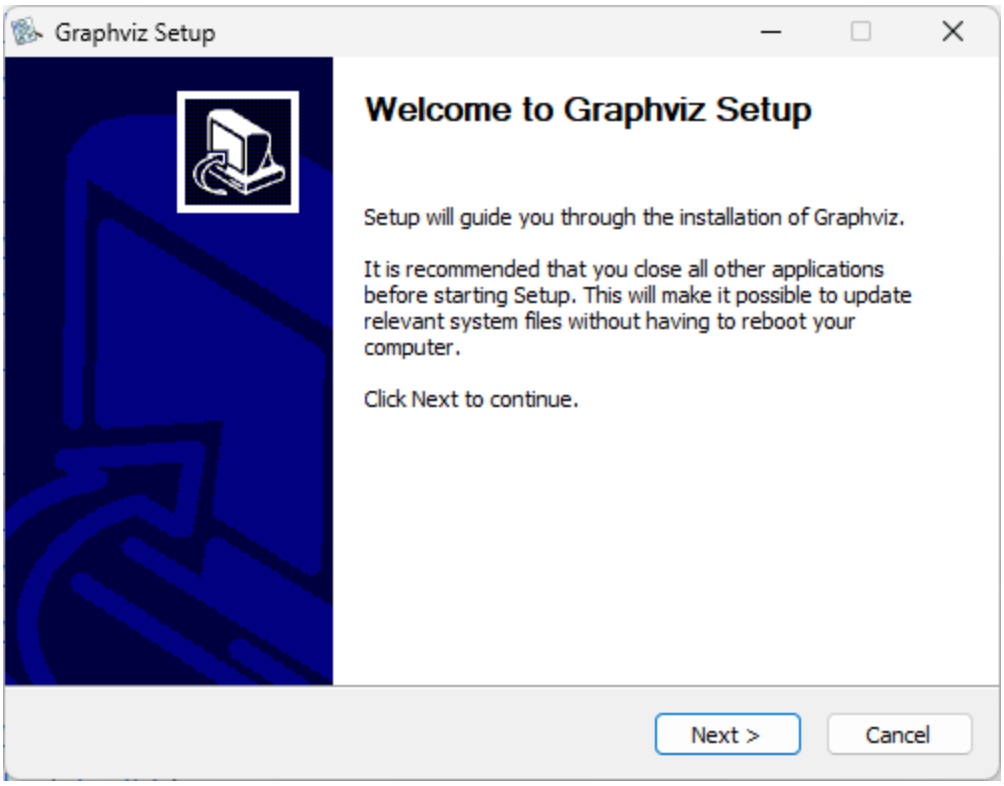
User Access Control
Do you want to allow this app from an unknown publisher to make changes to your device? windows_10_cmake_Release_graphviz-install-12.2.1-win64.exe Publisher unknown File origin: Hard drive on this computer Show more details

User Access Control
[Yes] [No]

Press the `Yes` button, and the Graphviz installer will begin to run.

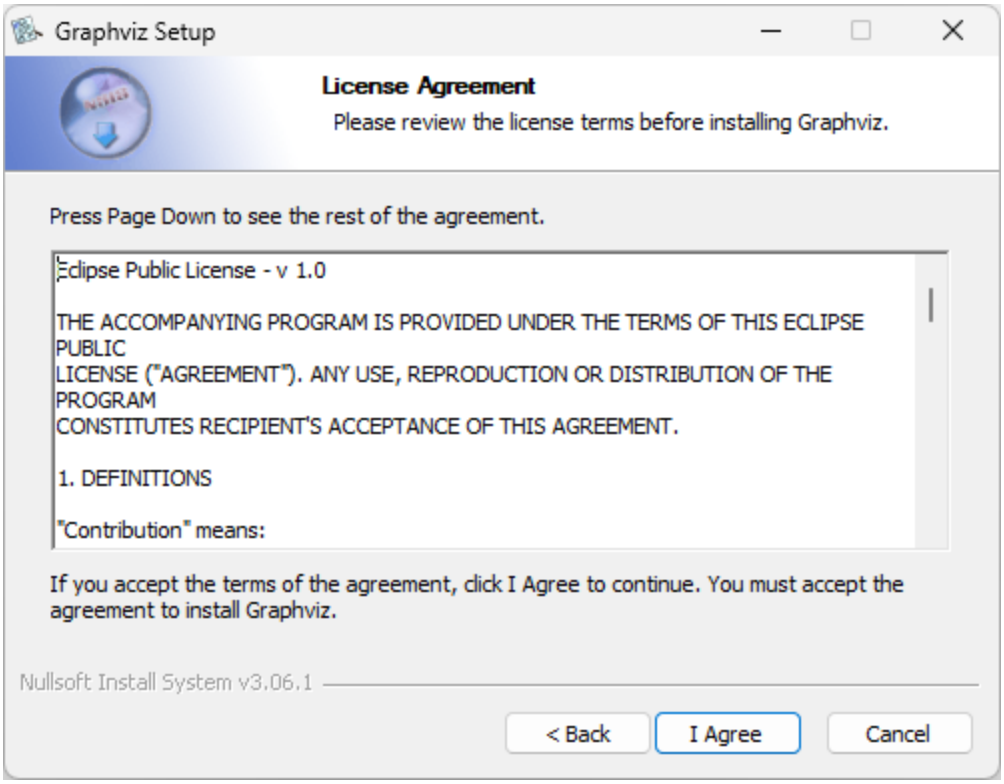
Step 3 - Select `Next >`

Select `Next >` on the **"Welcome to Graphviz Setup"** splash page.



Step 4 - Accept the License Agreement

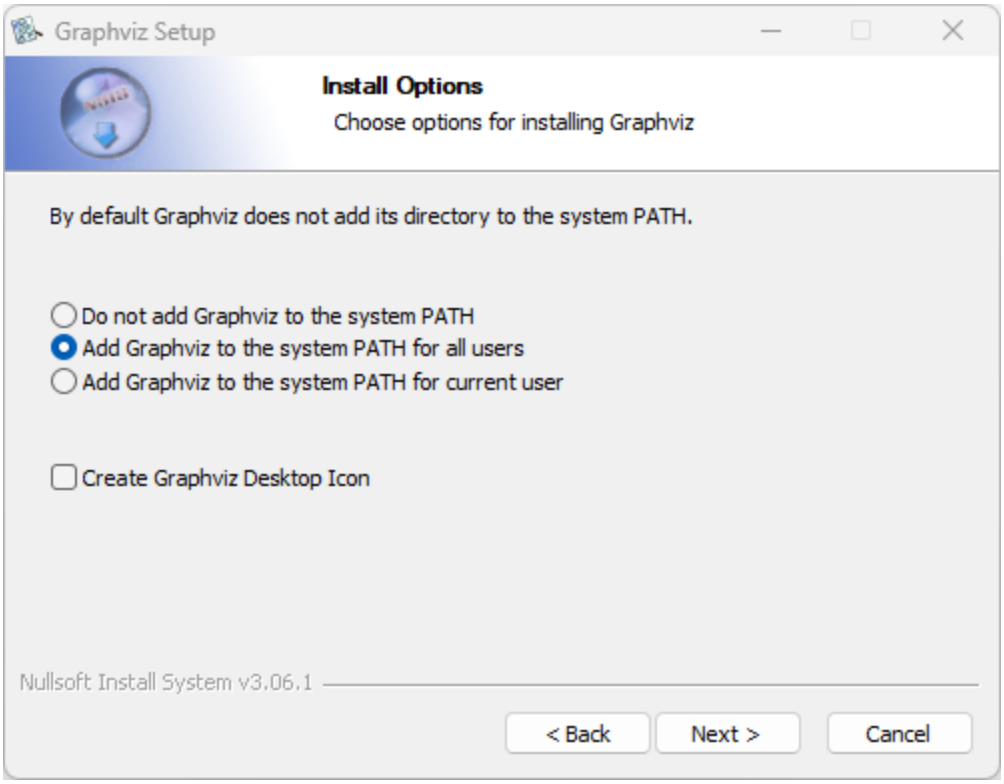
Accept the License Agreement by pressing the `I Agree` button.



Step 5 - Modify PATH

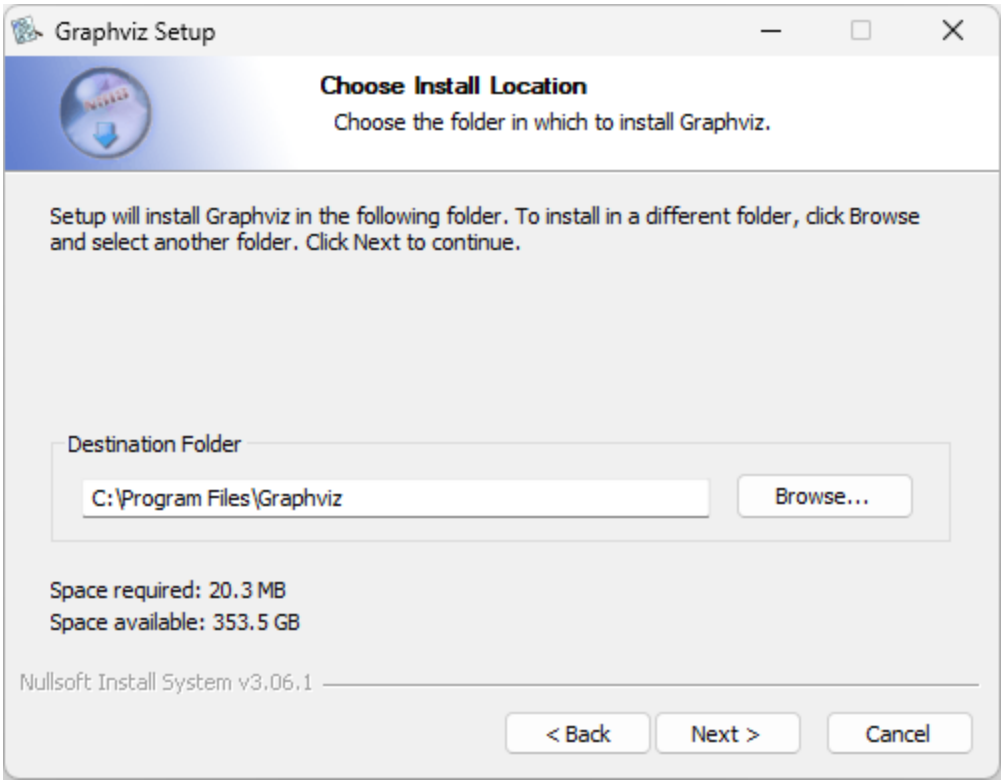
The Graphviz `bin` directory needs to be on your system PATH.

Select the radio button `Add Graphviz to the system PATH for all users`, then click the `Next >` button.



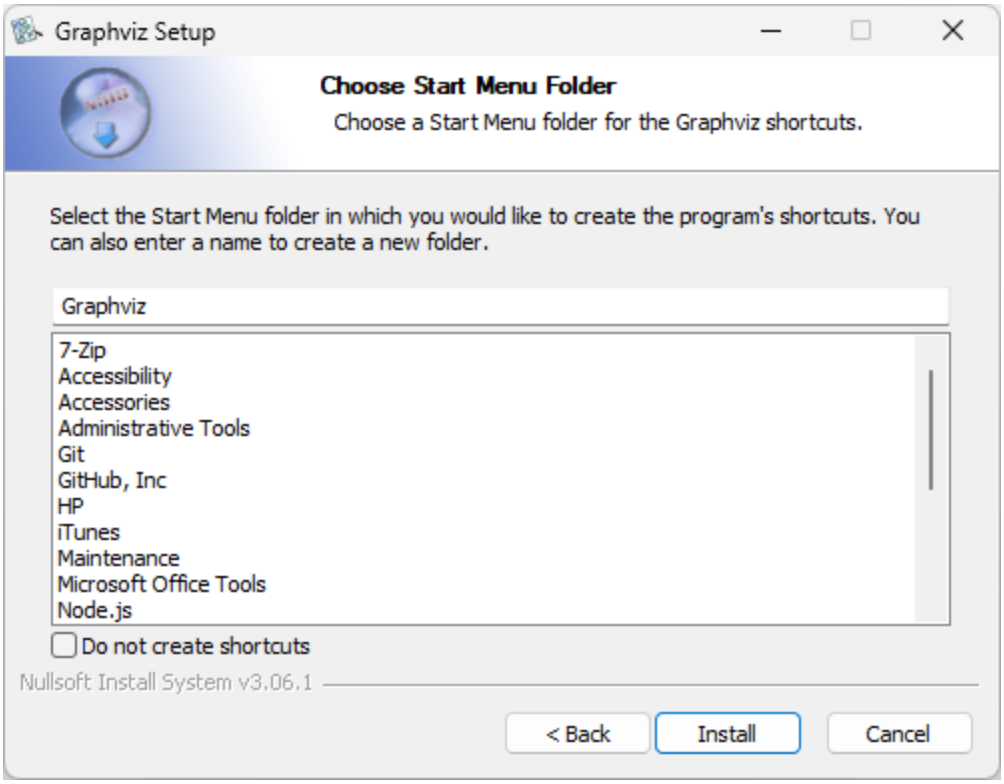
Step 6 - Specify Installation Folder

Specify the folder where Graphviz should be installed. Select the `Everyone` radio button, and then select the `Next >` button.



Step 7 - Press the `Install` button

Choose the Start Menu Folder and press the `Install` button.

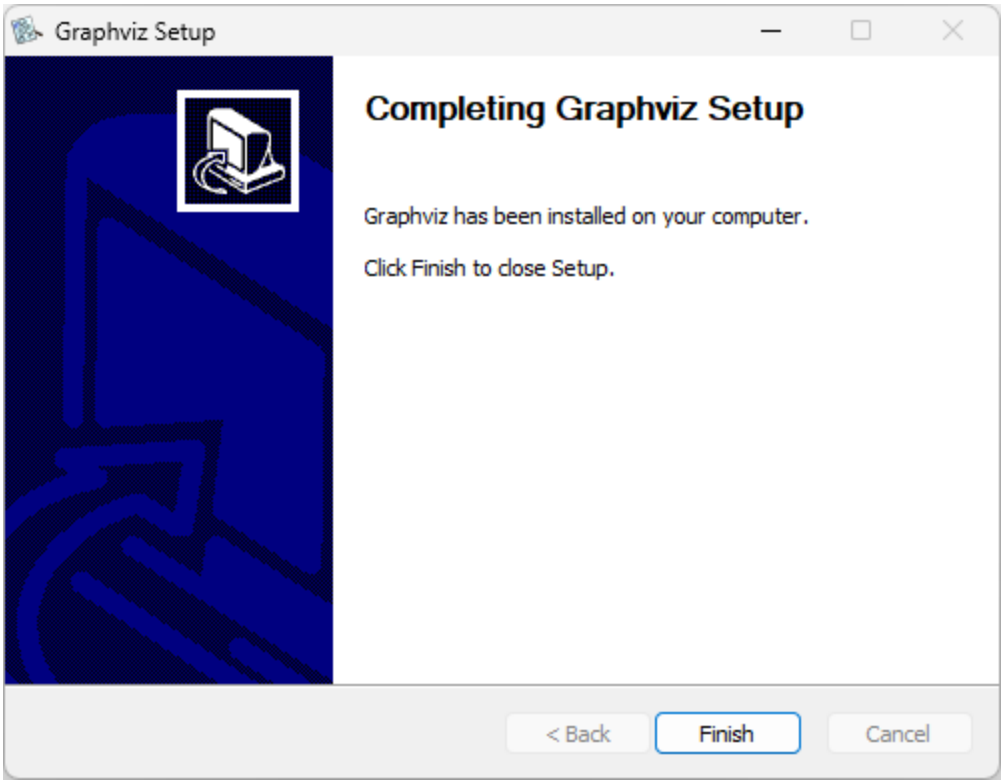


Step 8 - Monitor Progress

The installer will copy the files and make Windows configuration changes. A status bar will indicate how the installation is progressing.

Step 9 - Click `Finish`

Once the **Completing Graphviz Setup** screen appears press the `Finish` button. The software is installed.



Step 10 - Restart Microsoft Windows (Optional)

Technically, restarting Microsoft Windows shouldn't be necessary. However, if you've already been running Excel, it might be holding an outdated copy of your environment variables. Restarting Microsoft Windows ensures that Excel references the current `PATH` environment variable.

Open Command Prompt

At this point, you have completed the installation steps for the Graphviz software. Now perform a Graphviz Command Line configuration and test.

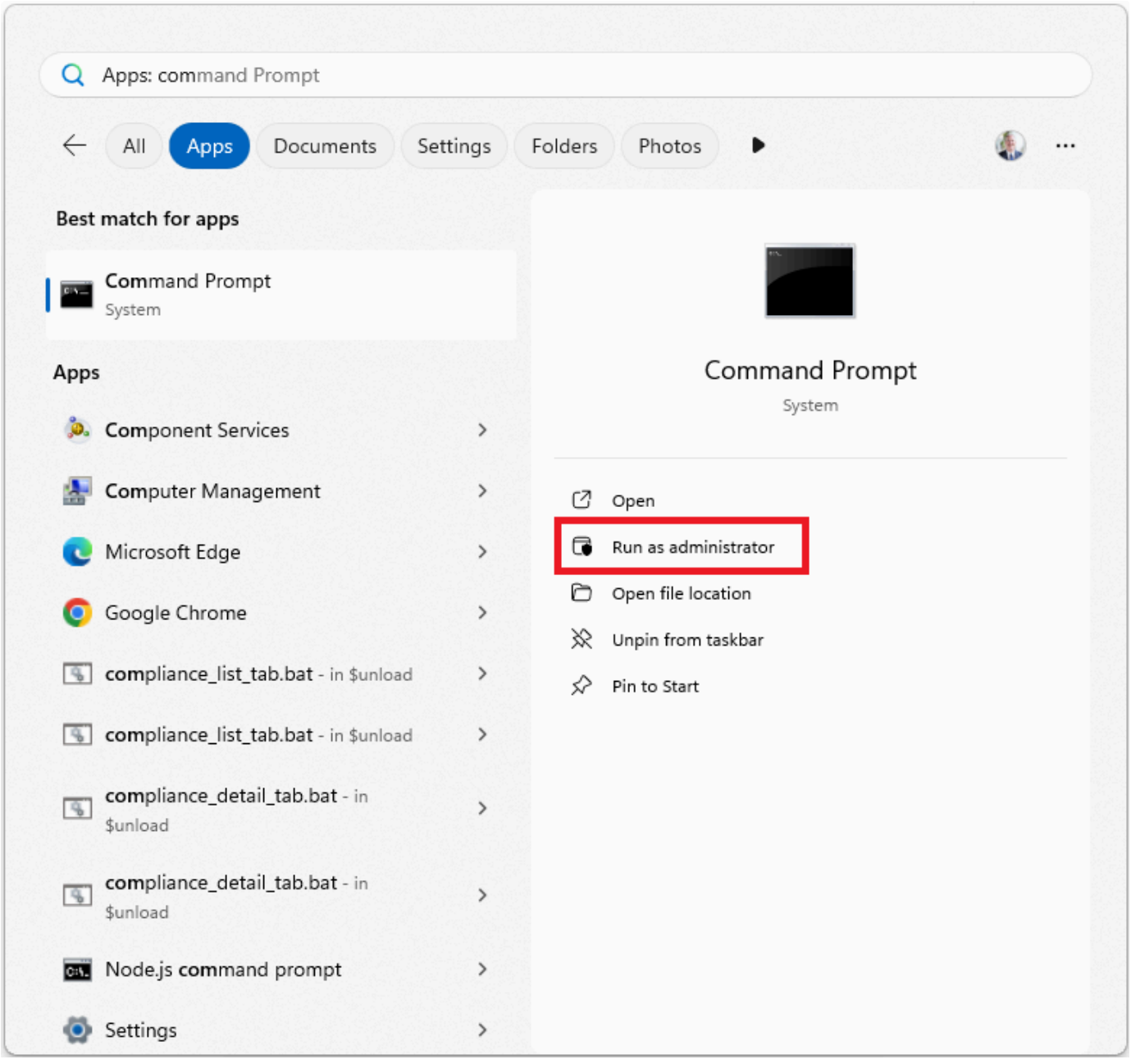
> **Do not skip these steps!**
>
> The Relationship Visualizer spreadsheet utilizes command-line programs to generate graph visualizations. You must manually execute a command-line command to configure the Graphviz plugins before using Graphviz properly.

Testing the command line programs prior to using the spreadsheet can help ensure that everything is in place correctly so that the spreadsheet can perform properly.

Step 1 - Open a Command Prompt

Open a Command Prompt window using the `Run as Administrator` option.

Click on the Windows Start Menu icon and begin to type `Command Prompt`. When the `Command Prompt` App appears choose the `Run as administrator` option.



Step 2 - Run as Administrator

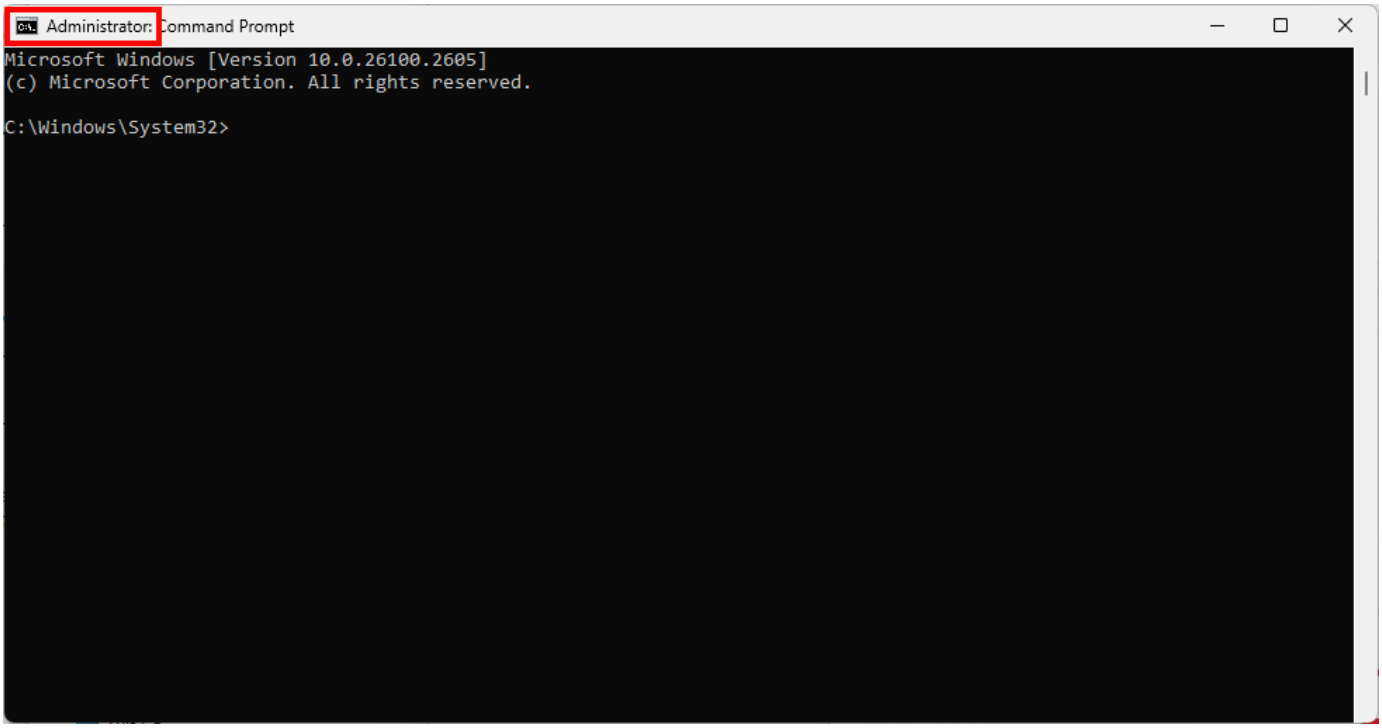
You will get asked for permission to run a command prompt as Administrator.

Press the `Yes` button.

> **User Access Control**
>
> Do you want to allow this app from an unknown publisher to make changes to your device?
> Windows Command Processor Verified Publisher: Microsoft Windows
>
> [Yes] [No]

Step 3 - Confirm Administrator rights

A command prompt window appears. Confirm that the word **Administrator** appears in the Window title.



Step 4 - Display Graphviz Version

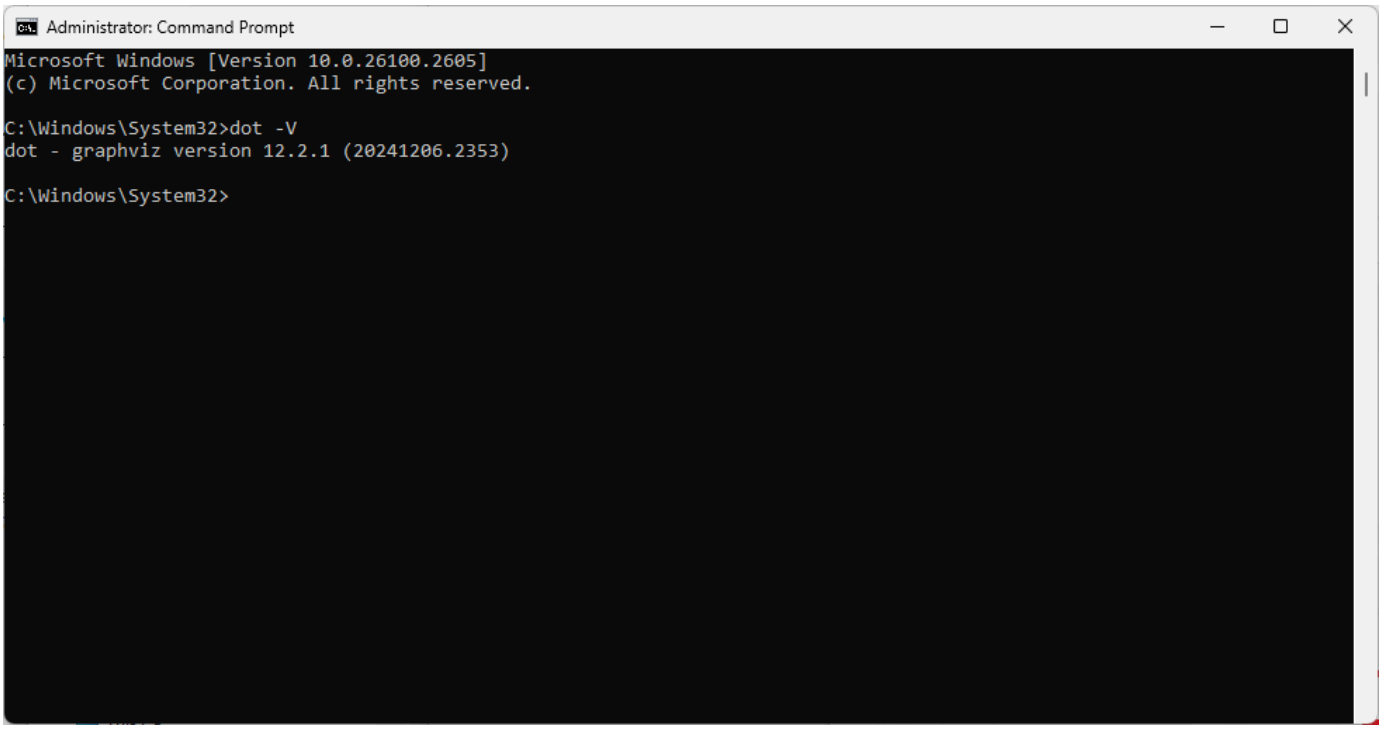
Test that Graphviz is configured properly on the PATH by entering the command:

```
dot -V
```

noting that the `-V` switch (for version) should be in uppercase, not lowercase. The dot program should respond with the message:

`dot - graphviz version 12.2.1 (20241206.2353)`

in similar fashion to the screen print below:



If you receive the message:

`'dot' is not recognized as an internal or external command, operable program or batch file.`

It means one of the following things:

- You have specified the path to the Graphviz executables incorrectly and you should repeat the steps above. Things to check are:
  - Did you remember to add the `bin` subdirectory to the Graphviz directory path?
  - Is the directory placed at the end of the `PATH` such that the length of the `PATH` exceeds the Windows length limit? If so, move the Graphviz bin directory closer to the beginning of the list.
- You opened the `Command Prompt` window prior to updating the `PATH` statement. This command window is still recognizing the old path. Close the `Command Prompt` window, open a new one, and repeat the `dot -V` command.

Step 5 - Configure Graphviz Plugins

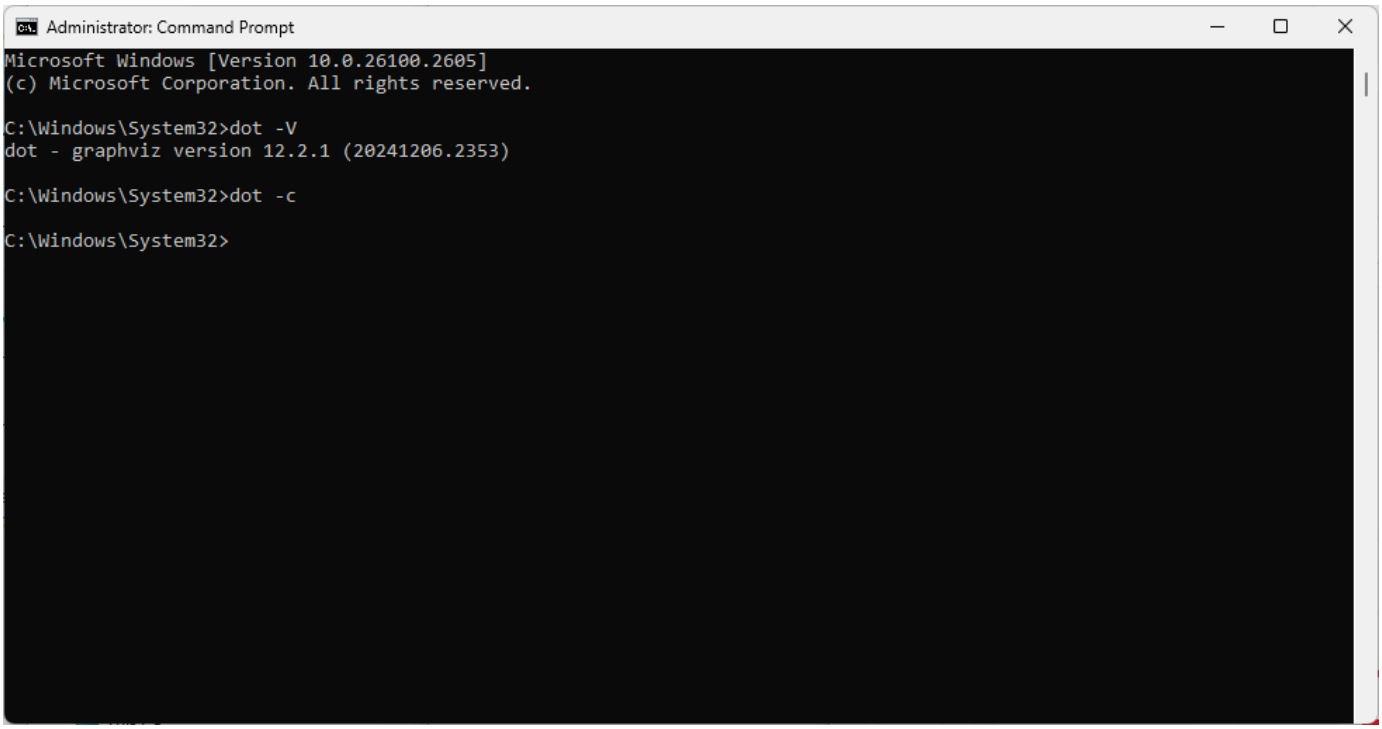
WARNING

Do not skip this important step!

Configure the plugins by entering the command

```
dot -c
```

No messages are written when the command executes; the screen will look as follows:

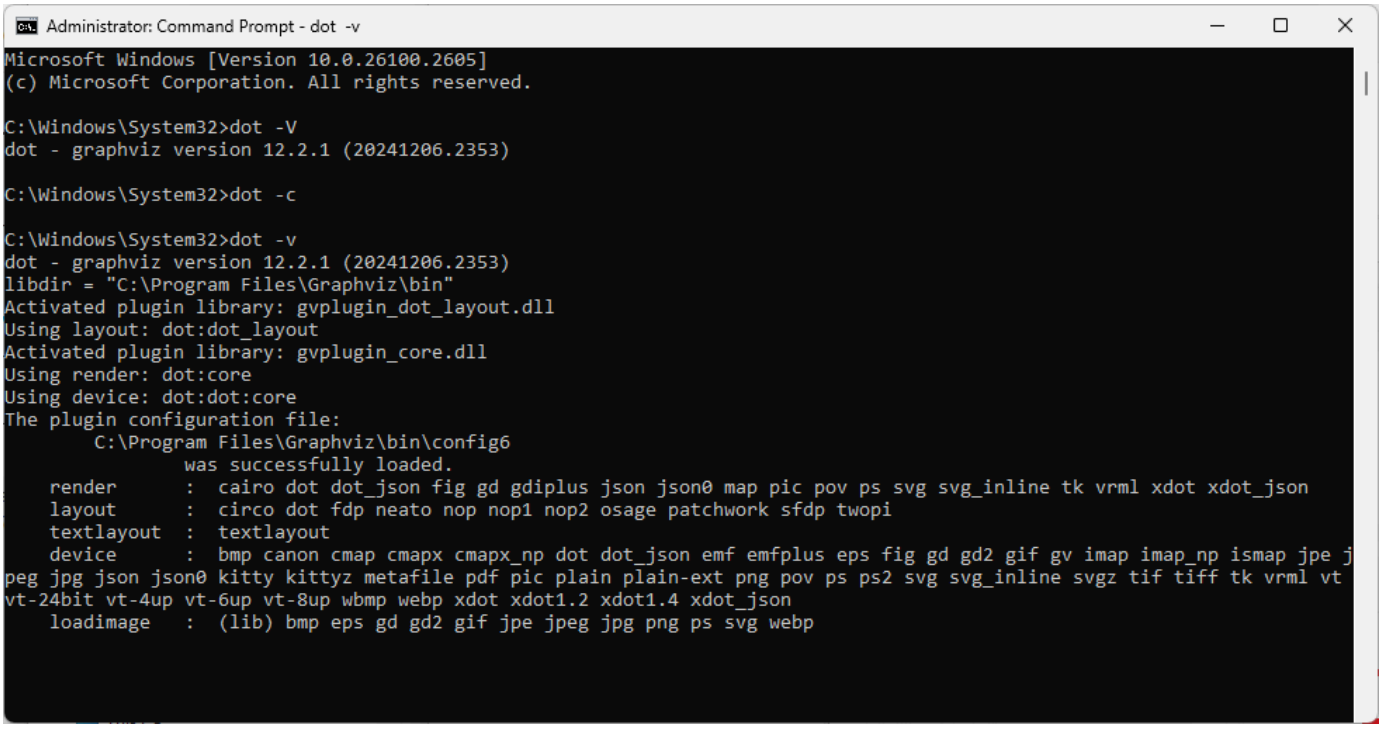


Step 6 - View the Plugins List

To see the list of configured plugins type the command

```
dot -v
```

where the `-v` is **lowercase**. The screen will appear as follows:



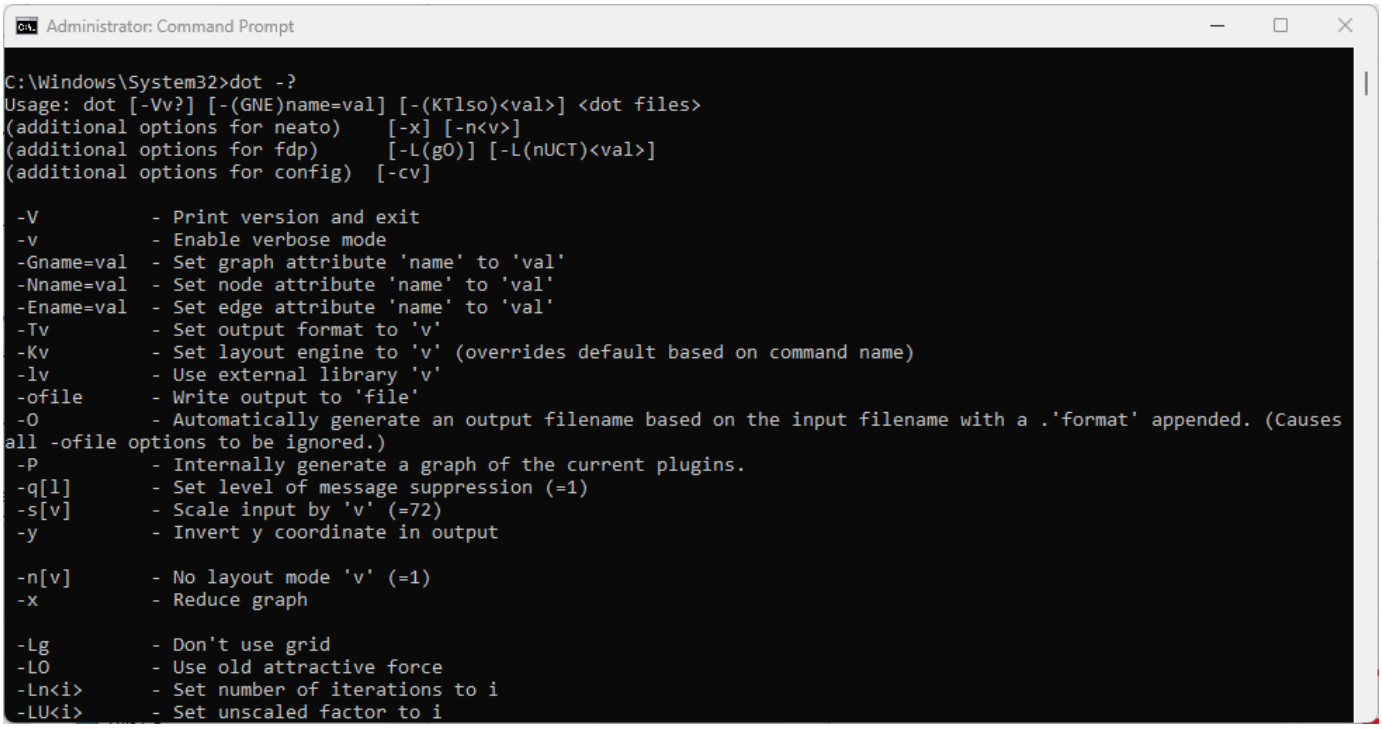
At this point Graphviz is waiting for more input. Pressing the `Ctrl` + `C` keys will break you from the `dot` program.

Step 7 - View Command Line Options

To see the list of command line options you can enter the command

`dot -?` or `dot --help`

The screen will appear as follows:

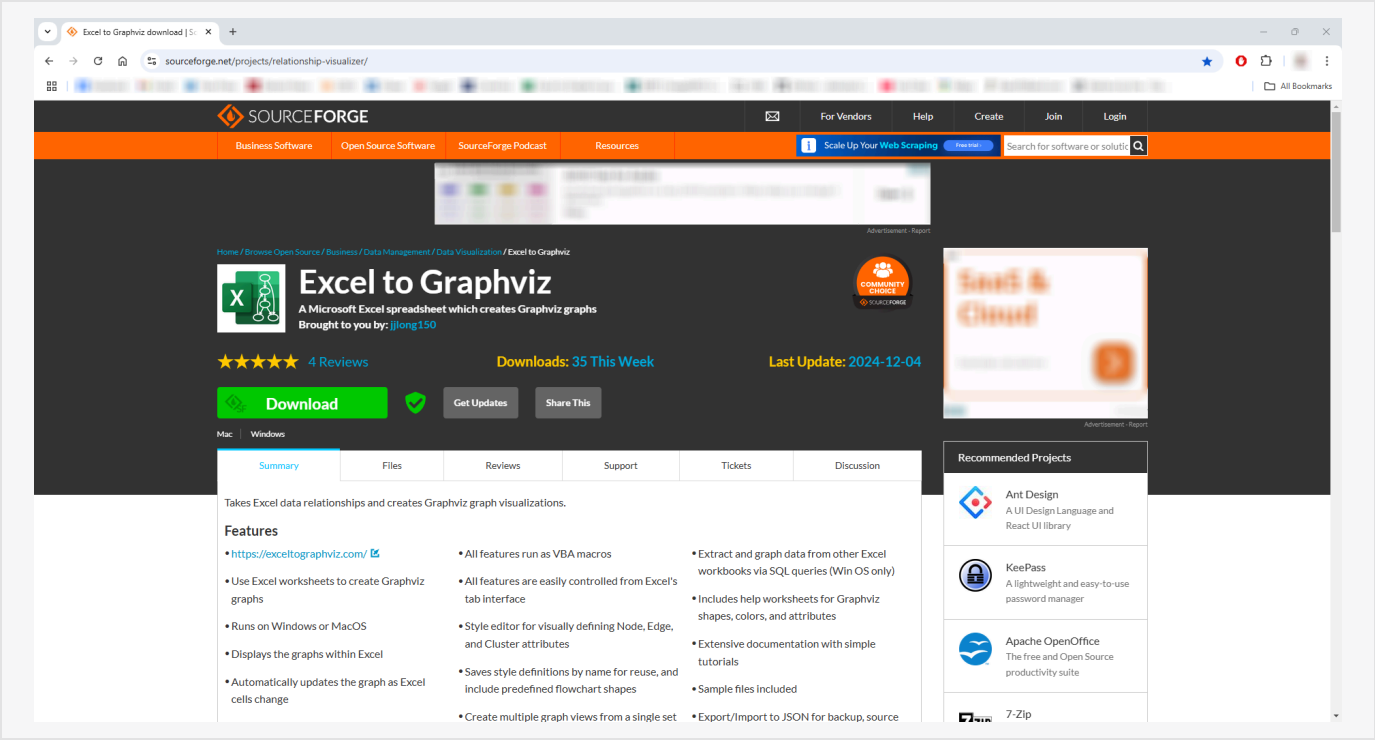


Congratulations! Graphviz is installed properly.

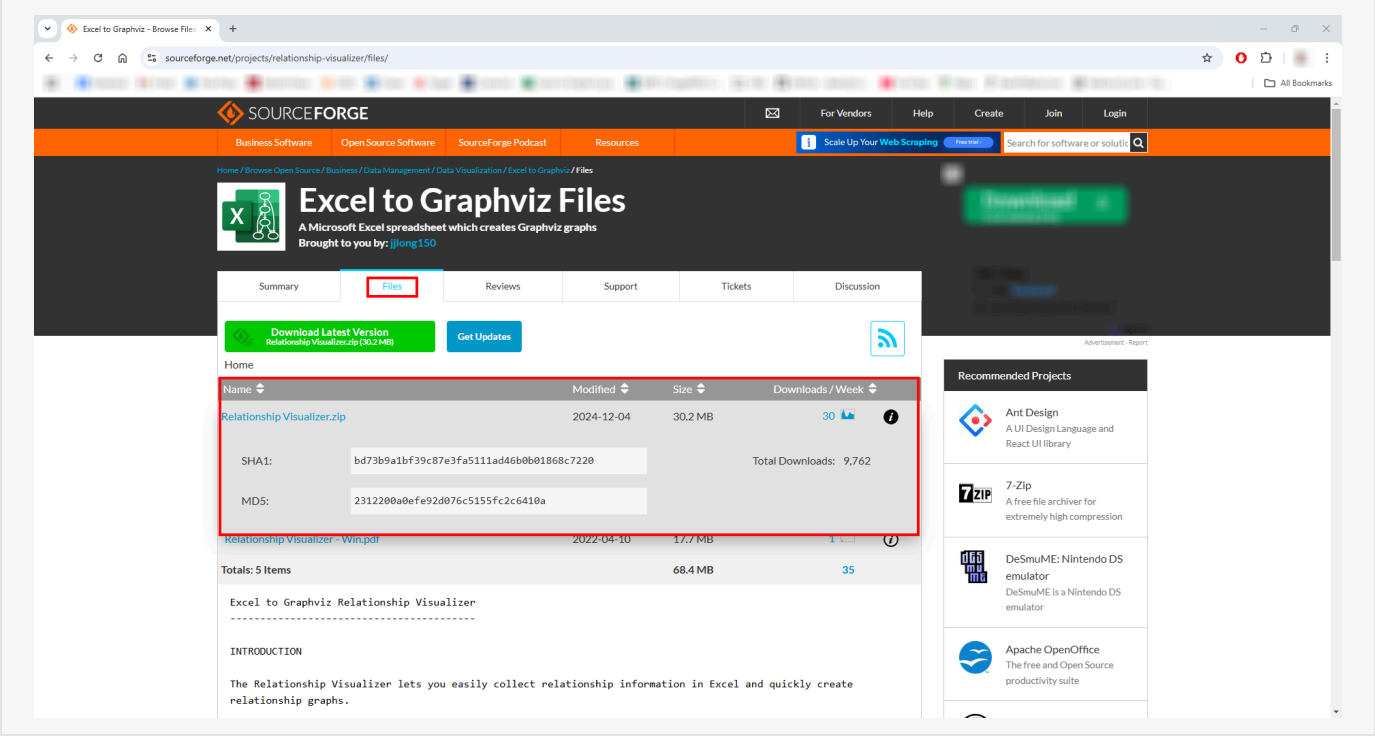
Download the Relationship Visualizer Assets

The *Relationship Visualizer* is distributed as a Zip file which is hosted at SourceForge, as SourceForge provides virus scanning of assets, and provides checksums to help confirm that files have not been tampered with during download.

Obtain `RelationshipVisualizer.zip` from SourceForge ↗ by clicking the large green `Download` button.

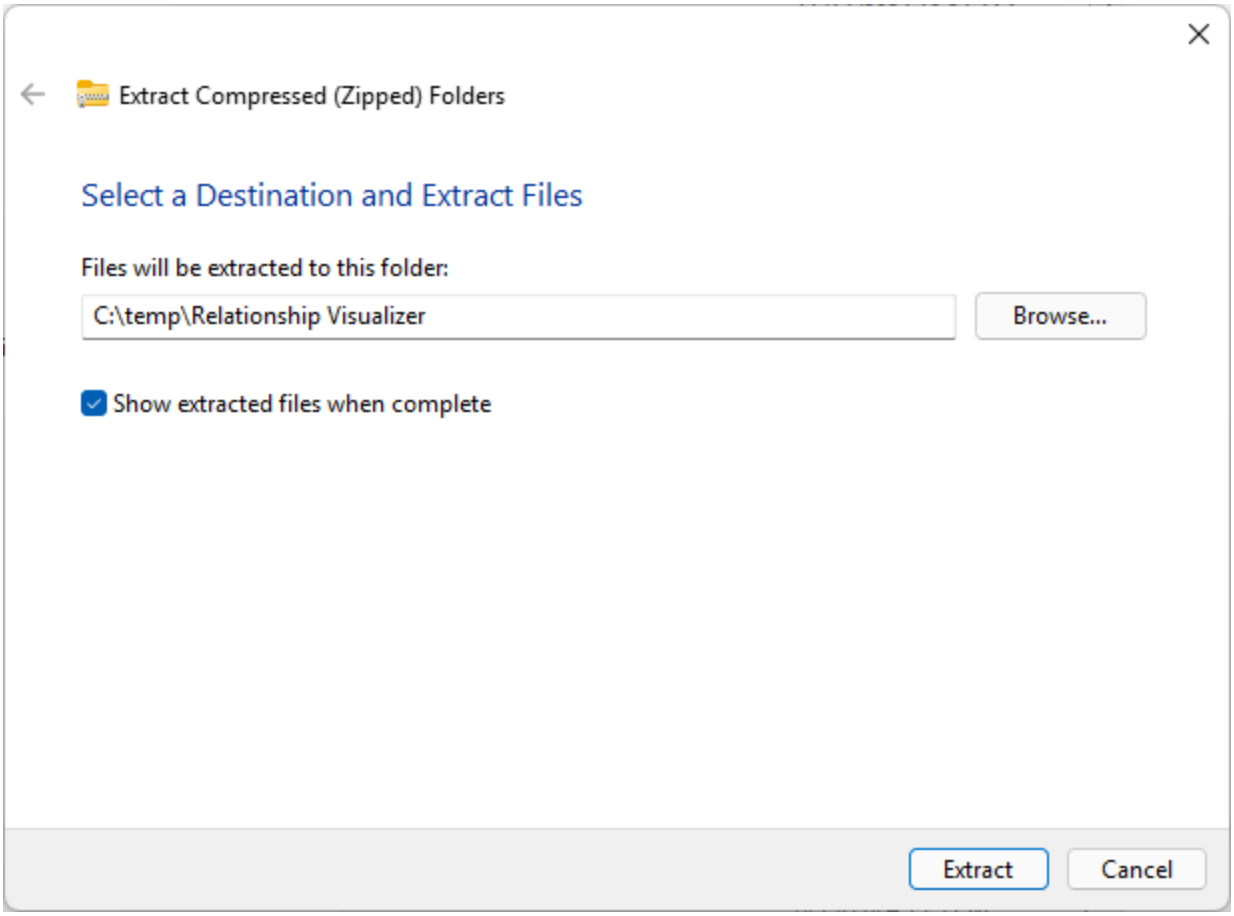


Optional, but recommended: Validate `SHA1` and/or `MD5` checksums available here ↗ against the file downloaded.



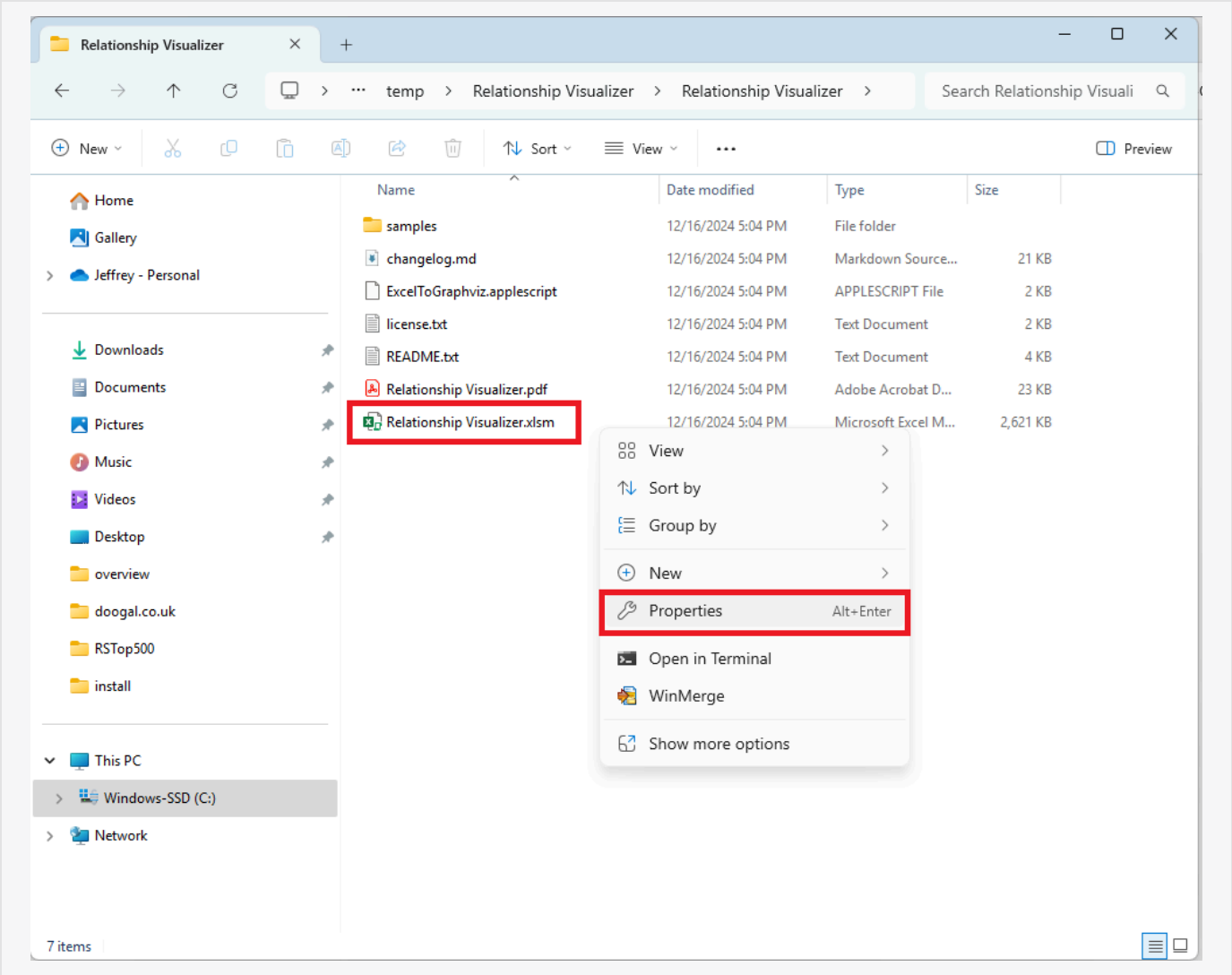
Extract Files From Zip File

Extract all files to a local directory.

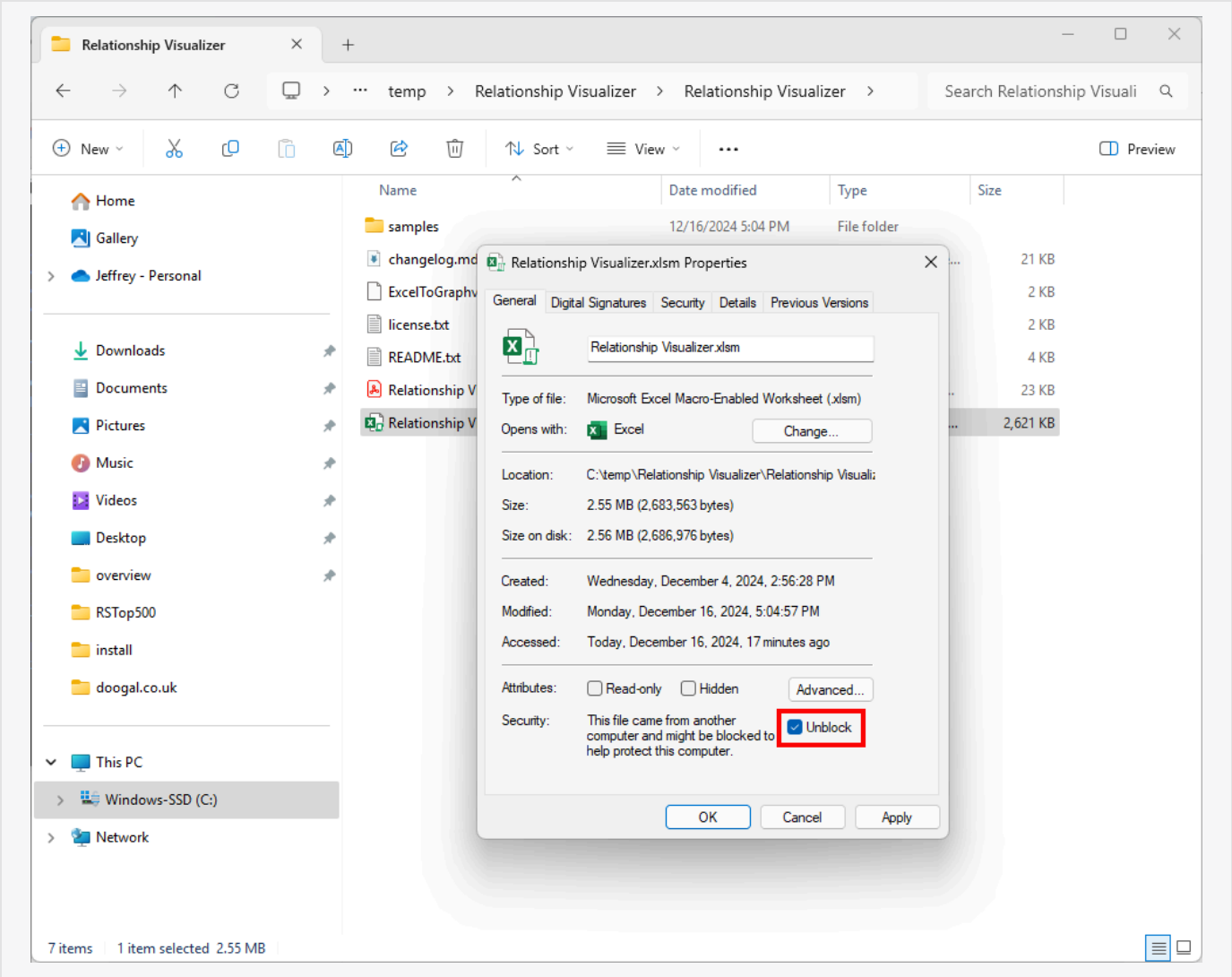


Unblock the Spreadsheet File

In the directory where you extracted the files, right mouse click (or `Alt+Enter`) on `Relationship Visualizer.xlsm` and select `Properties`.



Check `Unblock` at the bottom of the `Properties` dialog, click `Apply`, then click **OK**.

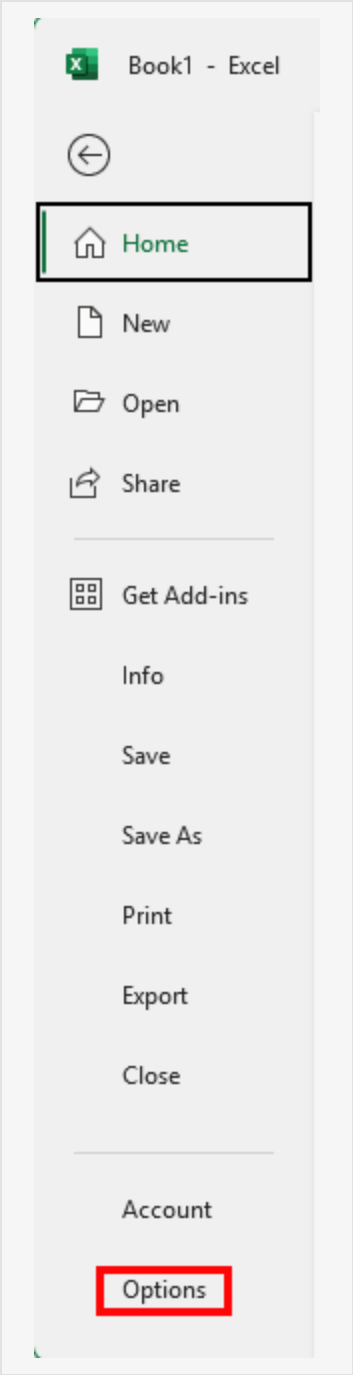


Open Microsoft Excel

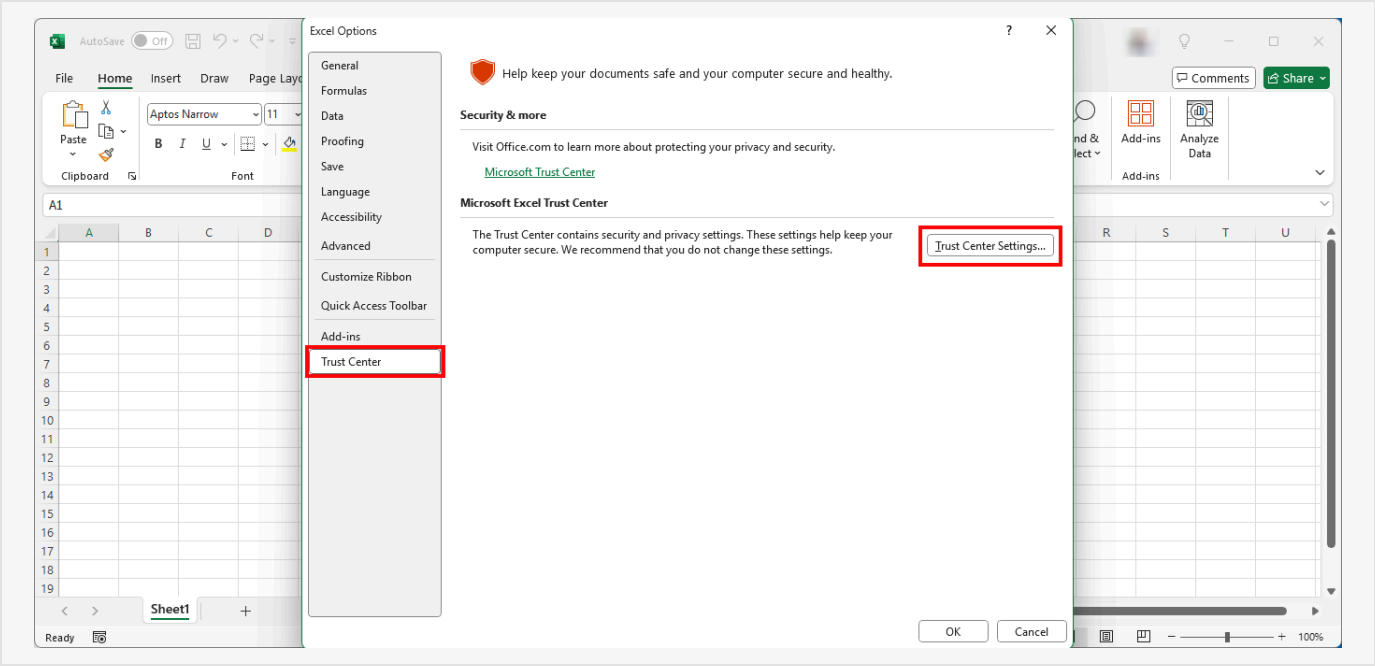
Step 1 - Enable Macros

The Relationship Visualizer is a VBA-enabled spreadsheet. If you do not trust VBA, or this download this tool is not for you. The Relationship Visualizer requires that Excel is configured to allow macros. You configure Microsoft Excel as follows:

From the `File` menu, select `Options` in the lower left corner.



The `Excel Options` dialog will be displayed. The left column has categories of options. Choose `Trust Center`, then press the `Trust Center Settings...` button.

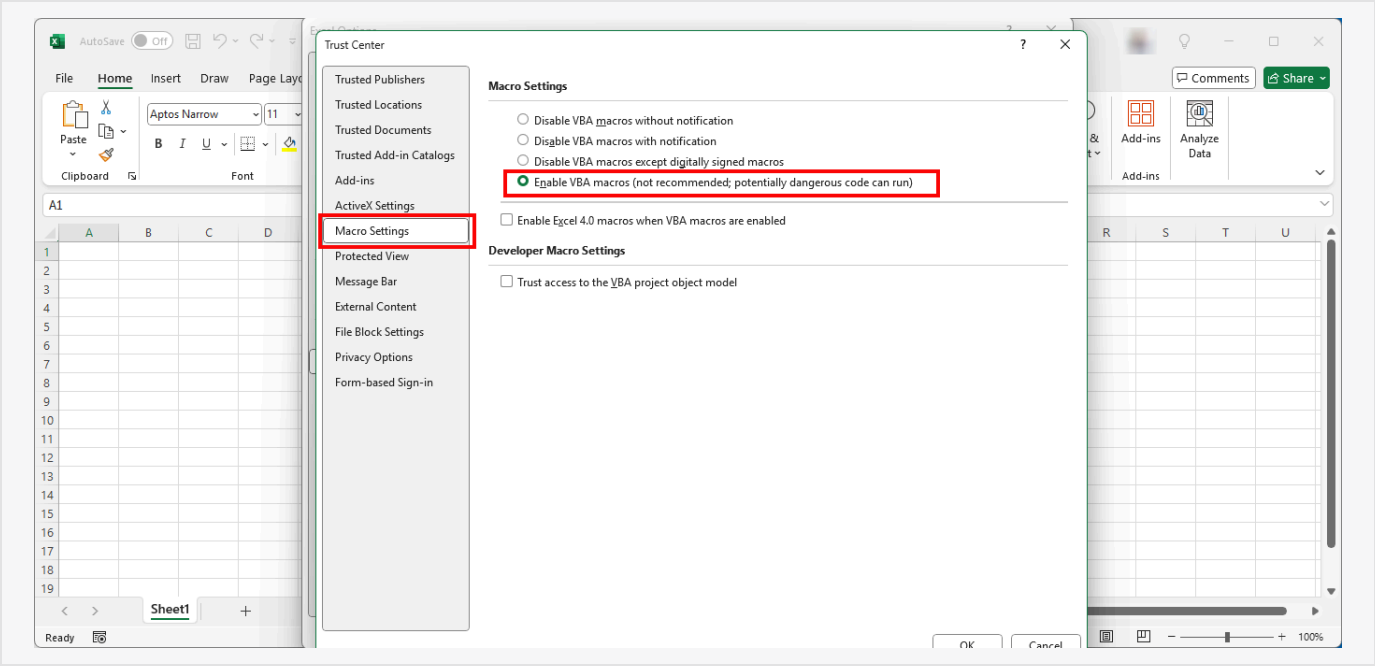


The left column list of categories will change to the Trust Center categories. Select `Macro Settings`.

Next, select the radio button which says `Enable VBA macros (not recommended; potentially dangerous code can run)`.

If you do not want to turn on unlimited macro access, you can choose the `

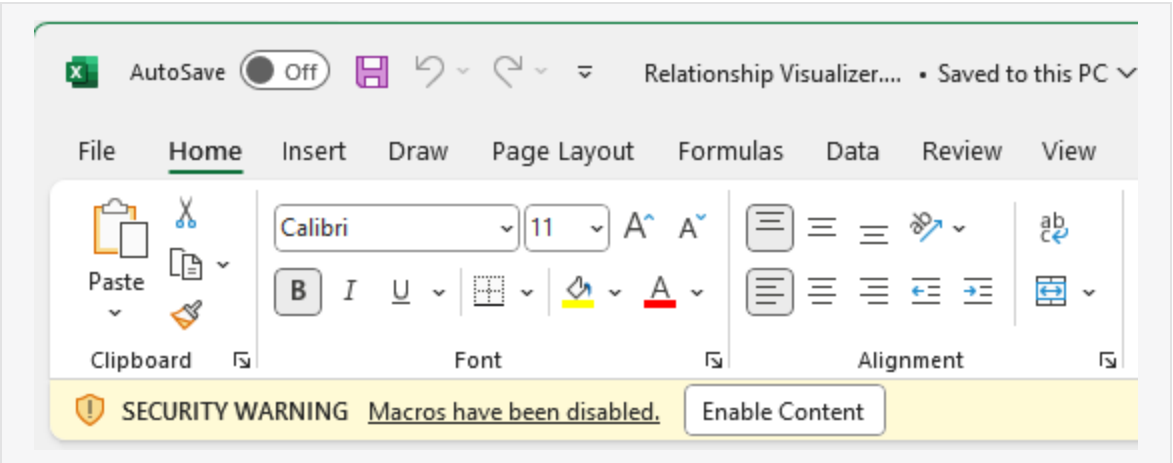
Click 'OK' to close the Trust Center Settings dialog, then click `OK` to close the `Excel Options` dialog.



Step 2 - Open Workbook

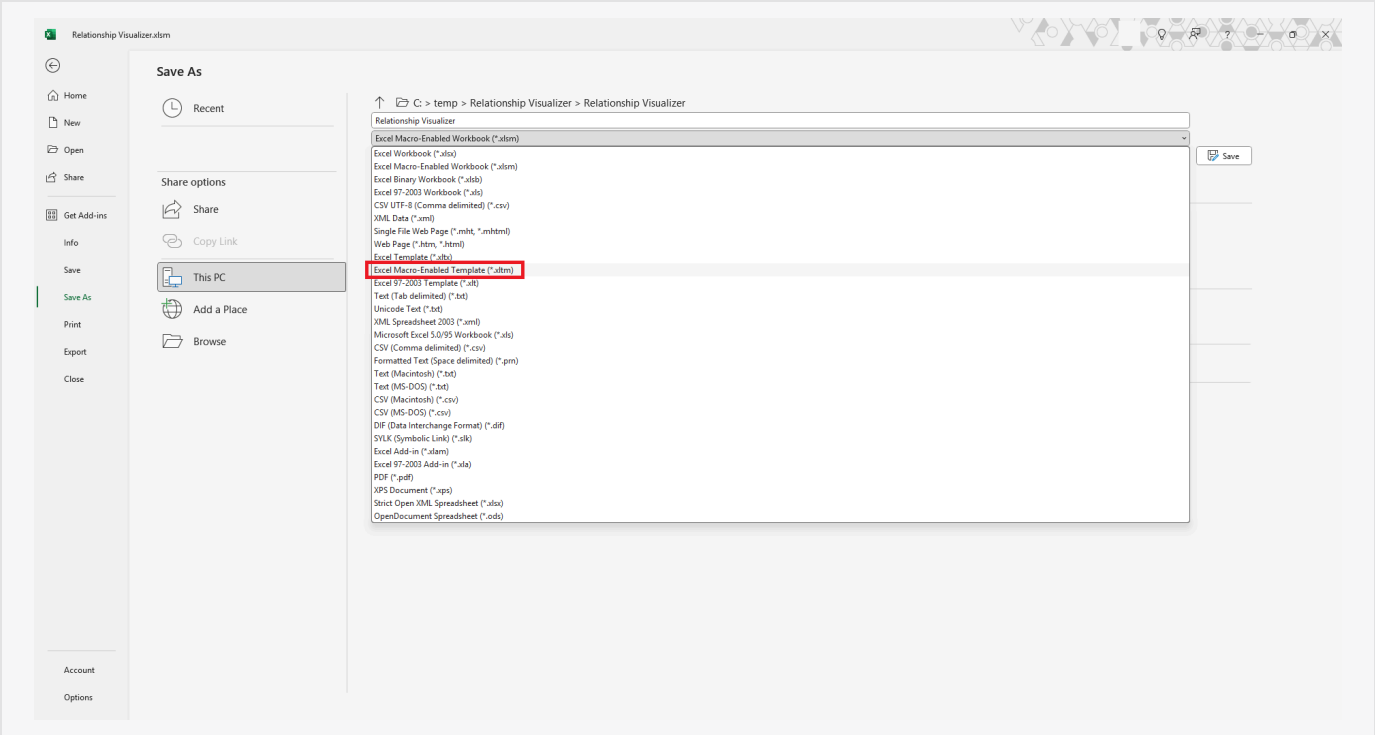
In the root directory of the Relationship Visualizer distribution there is a macro-enabled Excel spreadsheet named `Relationship Visualizer.xlsm`. Double-click the mouse on the file to launch Excel.

If you get the message `SECURITY WARNING Macros have been disabled [Enable Content]` click the `Enable Content` button. Enabling macro support is necessary to use the Relationship Visualizer spreadsheet.



Step 3 - Save the File as a Workbook Template

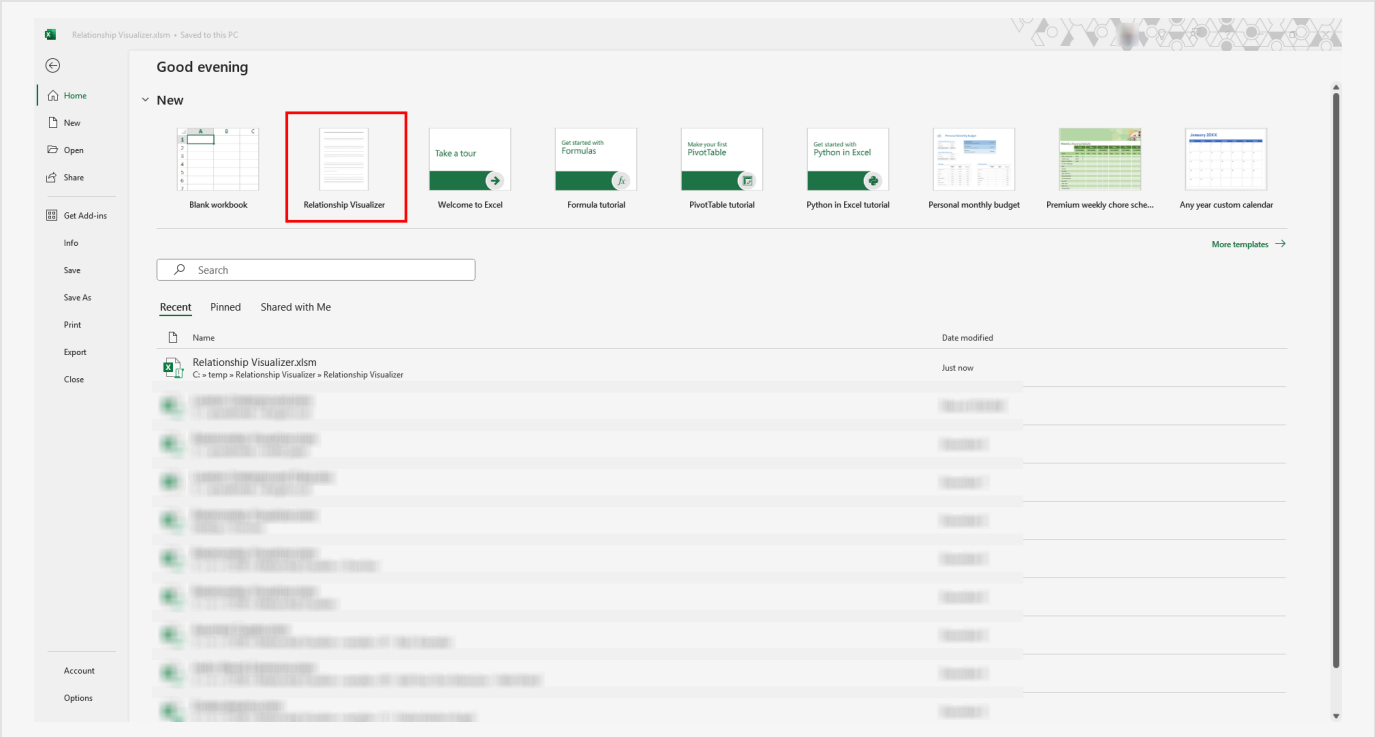
Perform a `File` → `Save As` operation.



Change the file extension from `Excel Macro-Enabled Workbook (*.xlsm)` to `Excel Macro-Enabled Template (*.xltm)`.

The file will be automatically stored in your custom Office templates directory (This PC > Documents > Custom Office Templates).

If you select `File` from the menu you will now see `Relationship Visualizer` as a choice. You can choose this template any time in the future when you want to create a fresh new copy of the Relationship Visualizer spreadsheet.


Relationship Visualizer.xlsm • Saved to this PC
Good evening
Home
New
Open
Share
Get Add-ins
Info
Save
Save As
Print
Export
Close
Account
Options
New
Blank workbook
Relationship Visualizer
Take a tour
Welcome to Excel
Get started with Formulas
Formula tutorial
Make your first PivotTable
PivotTable tutorial
Get started with Python in Excel
Python in Excel tutorial
Personal monthly budget
Premium weekly chore sche...
Any year custom calendar
More templates
Search
Recent
Pinned
Shared with Me
Name
Date modified
Relationship Visualizer.xlsm
C: » temp » Relationship Visualizer » Relationship Visualizer
Just now

macOS Installation Instructions

Brief Instructions

Perform these steps to install **Excel to Graphviz** on **macOS**

1. Use Homebrew ↗ to download and install **Graphviz** using the Install command:

   ```
   brew install Graphviz
   ```

2. Open a terminal window.

3. Run the command:

   ```
   sudo dot -c
   ```

   to register the Graphviz plugins.

4. Confirm Graphviz is working. Run the command `dot -V` to emit the version of Graphviz

5. Download file `RelationshipVisualizer.zip` from SourceForge ↗

6. Unzip file `RelationshipVisualizer.zip`

7. Determine where the `dot` command is installed. Issue the command:

   ```
   which dot
   ```

8. If `dot` is installed in `/usr/local/bin/dot`, then proceed with the next step, otherwise, edit the file

   ```
   ExcelToGraphviz.applescript
   ```

   and update the path

   ```
   /usr/local/bin/dot
   ```

at the top of the script to the match the path emitted by the `which dot` command.

9. Copy the file

   `ExcelToGraphviz.applescript`

   to folder

   `\~/Library/Application Scripts/com.microsoft.Excel`

10. Start **Excel**, and open the file

    `Relationship Visualizer.xlsm`

11. Enable macros, and grant permissions when prompted

12. Save the file as a template for creating future spreadsheets.

Detailed Instructions

Install Graphviz

The Graphviz Download Page ↗ offers 2 options for installing Graphviz, either by MacPorts or Homebrew.

I used homebrew to install Graphviz by following the instructions at this site https://formulae.brew.sh/formula/graphviz ↗ .

A video capture of a homebrew Graphviz installation can be viewed at https://www.youtube.com/watch?v=zRiUC82AnCk ↗

Configure Graphviz plugins.

This is an important step which must not be skipped. No messages are written when the command executes; the screen will look as follows:

You must have installer write privileges to configure the Graphviz plugins.

1. Open a `Terminal` window
2. Enter the command `sudo dot -c`
3. Enter the administrator password when prompted

Confirm Graphviz version

While you still have the terminal window open, issue the command `dot -V` (uppercase V).

If Graphviz is properly installed, it will emit a version number such as 2.44.1.

To see the list of configured plugins type the command

```
dot -v
```

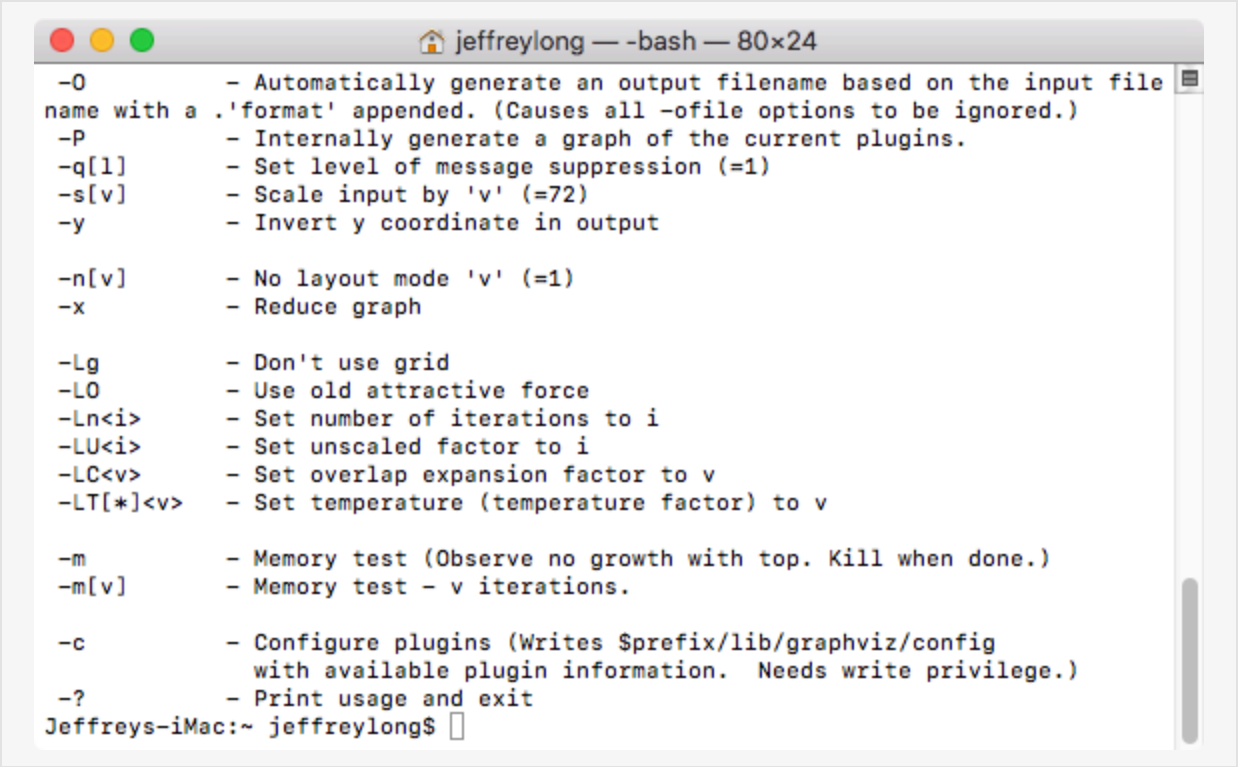
where the -v is lowercase. The screen will appear as follows:

At this point Graphviz is waiting for more input. Hitting the Command key + . (dot/period) key will break you from the dot program.

To see the list of command line options you can enter the command

```
dot -?
```

The screen will appear as follows:



Congratulations! Graphviz is installed properly.

Download file `RelationshipVisualizer.zip` from SourceForge.net

Excel to Graphviz is exclusively hosted on SourceForge.net. If you obtained a copy from any source other than direct download from SourceForge.net, then I suggest that you download the latest version from at https://sourceforge.net/projects/relationship-visualizer/ ↗

Unzip file `RelationshipVisualizer.zip`

The contents of “RelationshipVisualizer.zip” may be stored in any location. The zip file contains the macro-enabled spreadsheet “Relationship Visualizer.xlsm”, the corresponding Apple Script file ExcelToGraphviz.applescript, user documentation, samples, and license files.

The file ExcelToGraphviz.applescript must be installed in a specific location per Microsoft’s sandbox rules. This location is explained in future steps.

Edit file `ExcelToGraphviz.applescript`

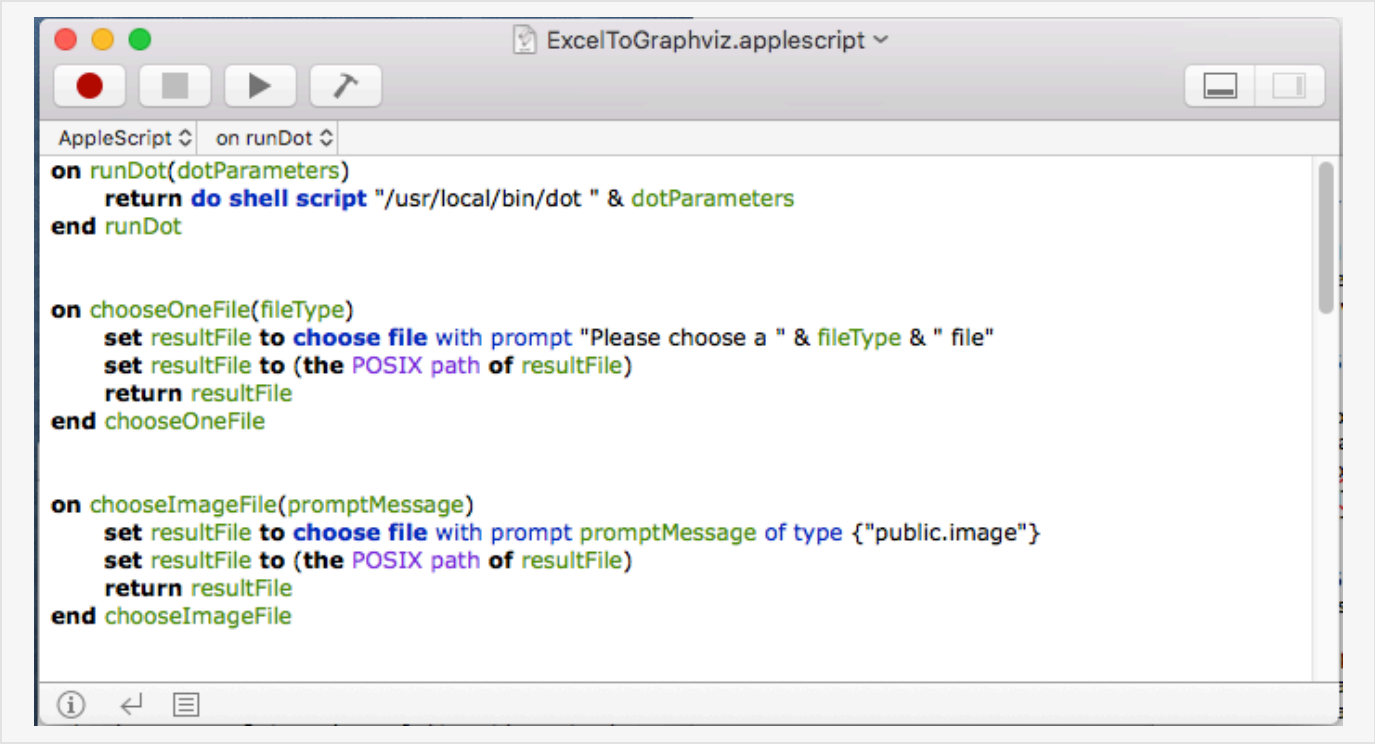
Update the path to the Graphviz `dot` command in the file `ExcelToGraphviz.applescript` (if necessary).

Homebrew installs the dot command in the folder `/usr/local/bin`. You can enter the command

```
which dot
```

in the terminal window to see where dot is installed.

If you get a response other than `/usr/local/bin/dot`, then you must edit the file `ExcelToGraphviz.applescript` and change the path in the command on line 2 from `/usr/local/bin/dot` to the path where dot is installed on your Mac.

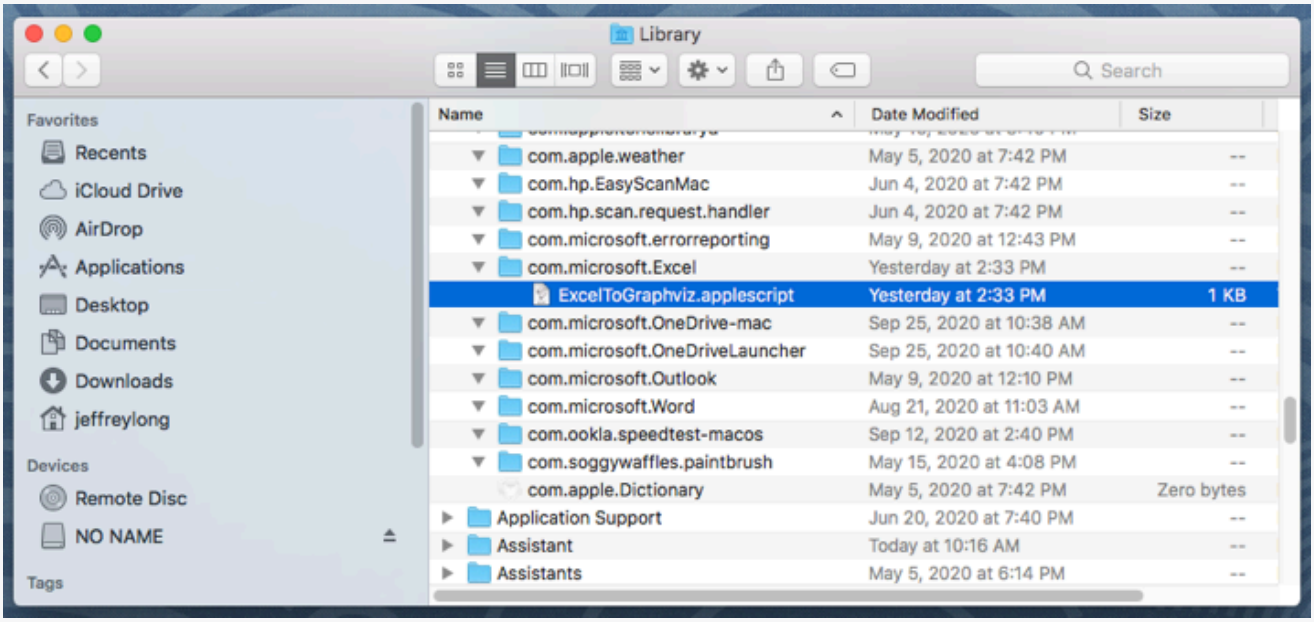


Copy Applescript script to sandbox folder

Copy file `ExcelToGraphviz.applescript` to folder `~/Library/Application Scripts/com.microsoft.Excel`

Now the script file is ready and tested we must copy it into the correct location. Follow the steps below to copy and paste it into this exact location.

1. Open a Finder Window
2. Hold the `Alt` key and click `Go` in the Finder menu bar
3. Click `Library`
4. Click `Application Scripts` (if it exists; if not create this folder)
5. Click `com.microsoft.Excel` if it exists; if not create this folder (note: Capital letter `E`)
6. Copy the file `ExcelToGraphviz.applescript` to the folder `com.microsoft.Excel`



Microsoft Excel Sandboxing Explained

Unlike prior versions of Office apps that support VBA, Office 2016 for Mac apps are sandboxed. Sandboxing restricts the apps from accessing resources outside the app container. This affects any add-ins or macros that involve file access or communication across processes.

Earlier versions of Office for Mac included a command called `MacScript` that supported inline AppleScripts. Although that command still exists in Office 2016 for Mac, `MacScript` is deprecated and its powers have been reduced. The `MacScript` command cannot invoke other applications, such as Finder, in Office 2016 for Mac due to the new sandbox rules.

Microsoft added a new VBA command `AppleScriptTask` that accesses and runs an `AppleScript` file located outside the sandboxed app. This new approach is not as convenient:

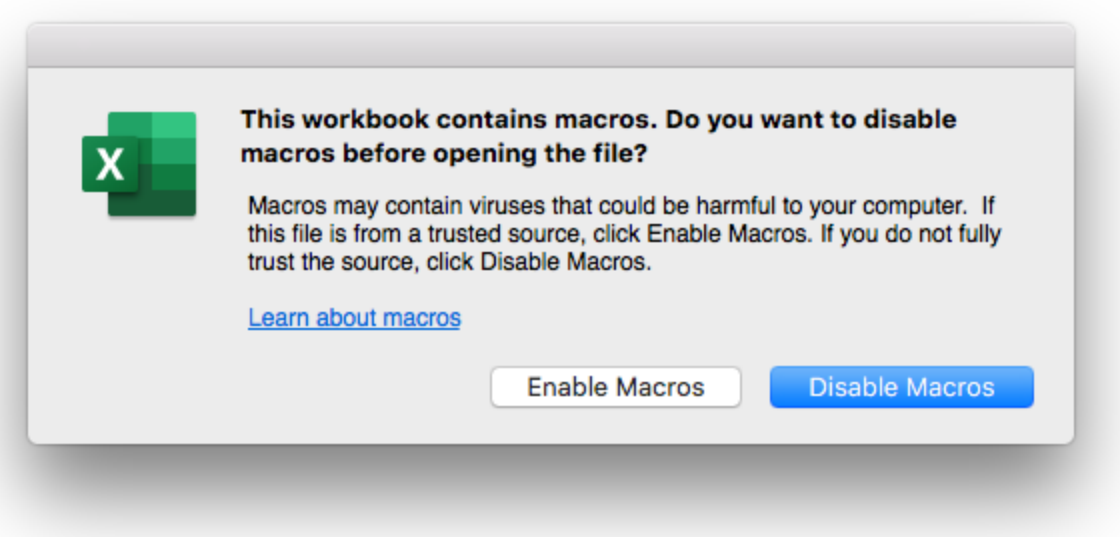
1. With the `MacScript` function the commands needed to run Graphviz's dot command could reside in the Excel file itself, while with the `AppleScriptTask` method it has become necessary to distribute an `AppleScript` script file.
2. Furthermore, this `AppleScript` file must be placed in a folder location specified by Microsoft's sandbox rules on the user's system to have permission to run. This sandbox requirement requires a user interaction the first time to place the script in the required folder location `~/Library/Application Scripts/com.microsoft.Excel`

Open the file `Relationship Visualizer.xlsm`

Open the file `Relationship Visualizer.xlsm` in Excel by double clicking on the file `Relationship Visualizer.xlsm` provided in the zip file

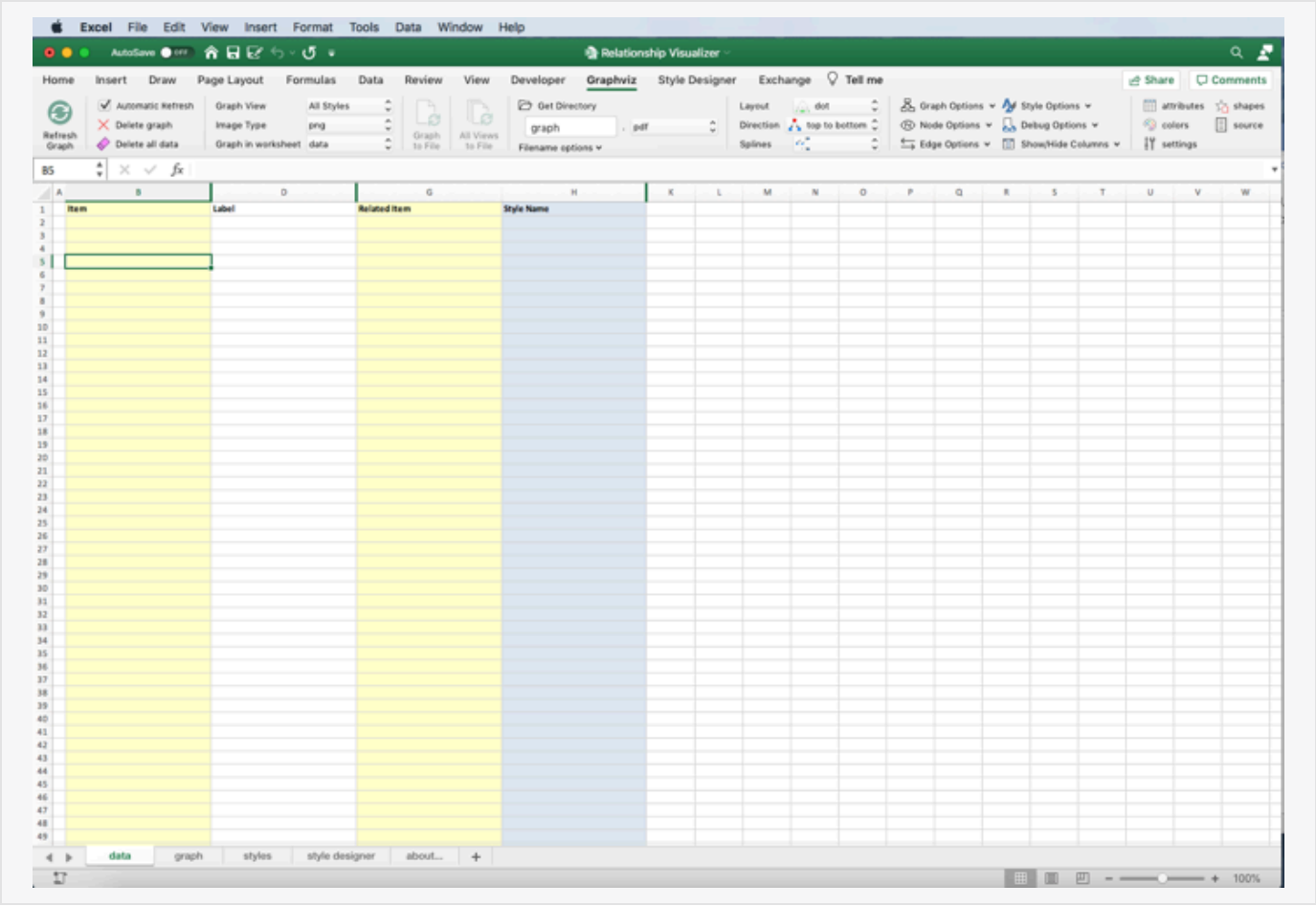
Enable Macros

Excel to Graphviz performs its work using VBA macros. When you launch the file, you will receive warnings of the macros and must give permission for the macros to run.



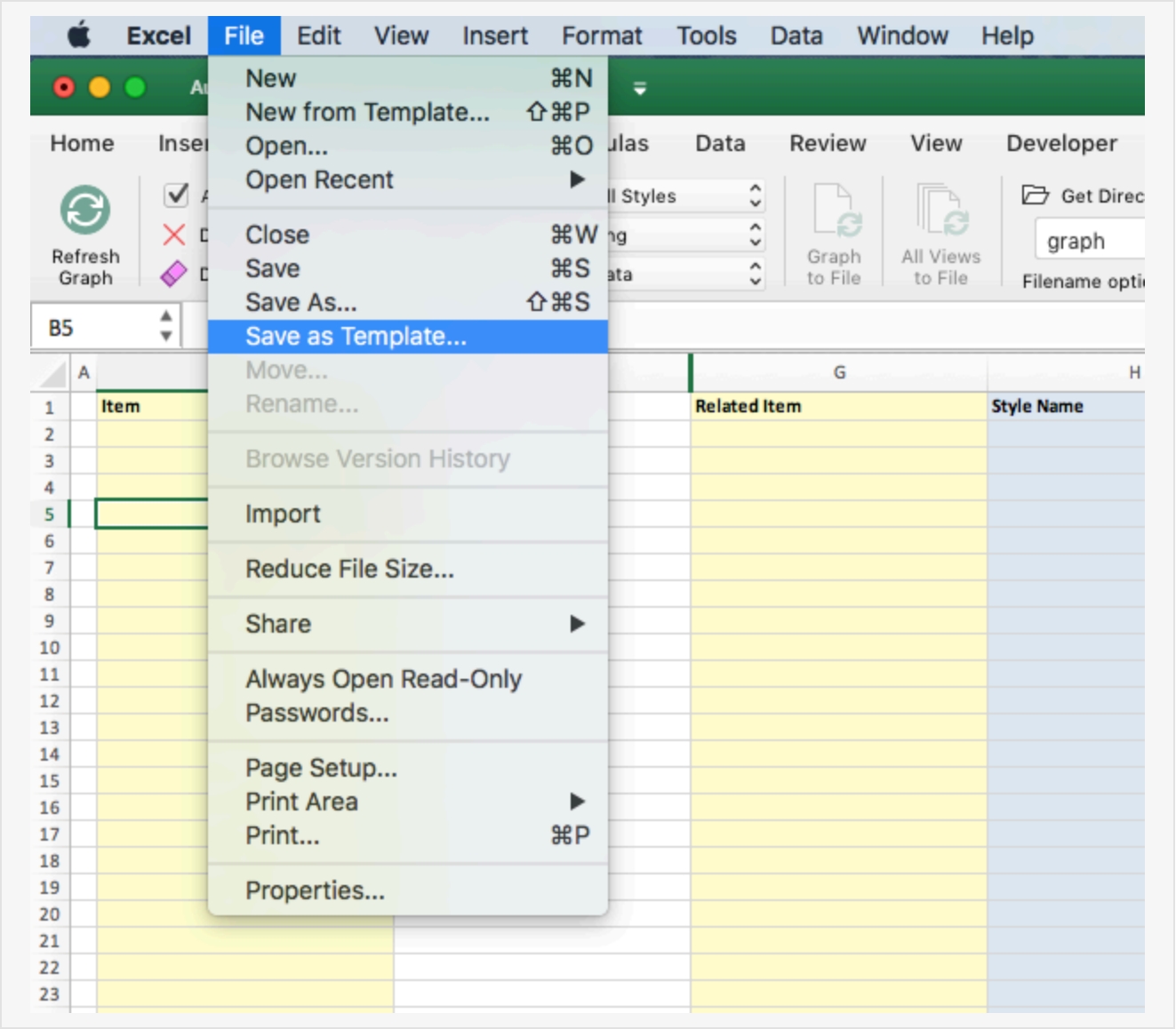
Save `Relationship Visualizer.xlsm` as a template.

Once you enable the macros, the full spreadsheet appears as:

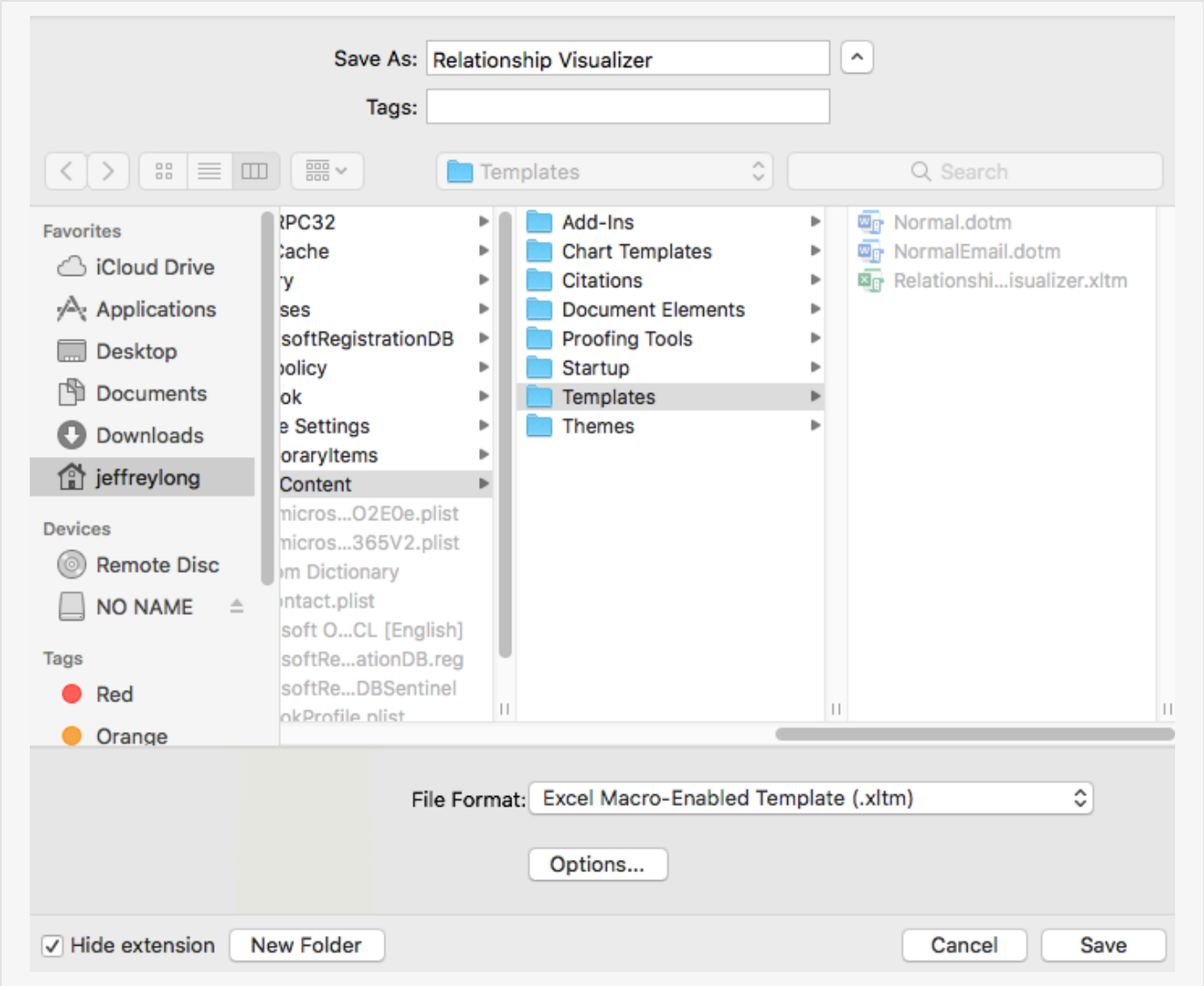


Perform a "File → Save As" operation. Save the workbook as a template so you can use the template to create more workbooks instead of starting from scratch. A template allows you to avoid having to make a copy of a file and clearing out old content.

From the `File` menu, select `Save as Template…`



When the file save dialog appears, provide a “Save As” name (e.g., Relationship Visualizer) and specify the file format as “Excel Macro-Enabled Template (.xltm)”.



Note that Excel automatically specifies the Microsoft Office Templates directory.

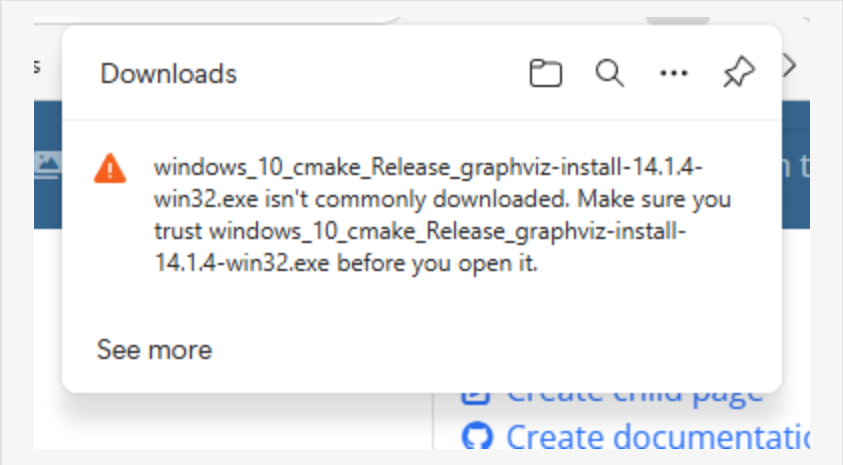
Microsoft Edge has Blocked the Graphviz Download

The Microsoft Edge browser may blocked the download of the Graphviz installer file.

If blocked, you will see an orange `/!\` warning icon in the download button.

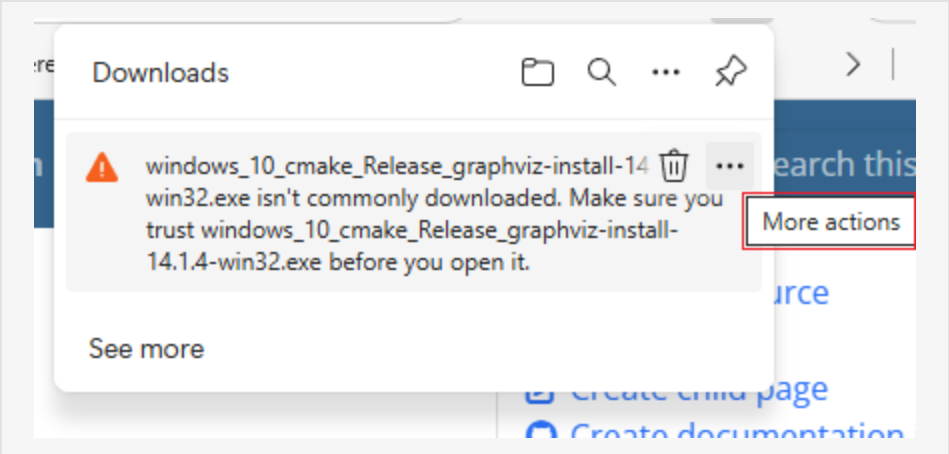
Step 1 - View Warning Message

Clicking on the button provides the reason the download was blocked.



Step 2 - Make `More Actions` appear

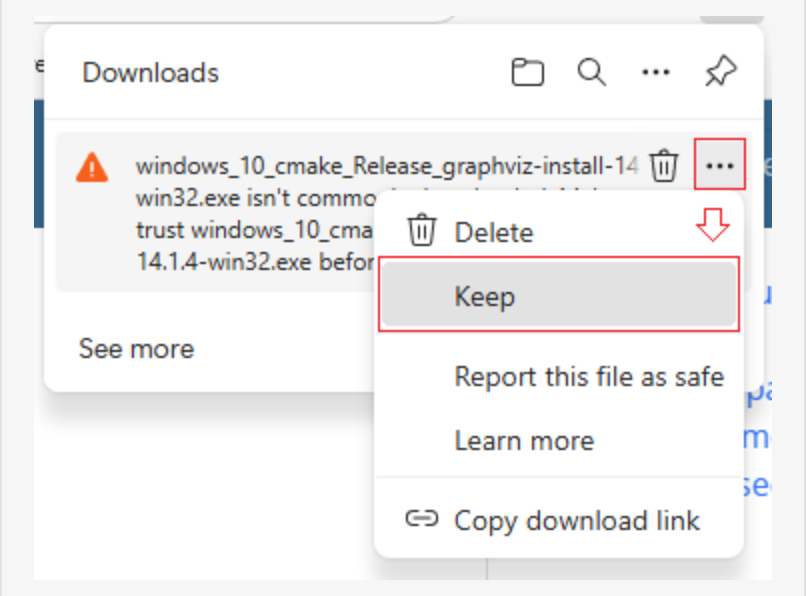
Hover your mouse over the message, and a button with three dots will appear, along with a tooltip which says, `More actions`.



Click on the `···` button, and a popup menu appears.

Step 3 - Select `Keep`

Select `Keep` from the dropdown list.

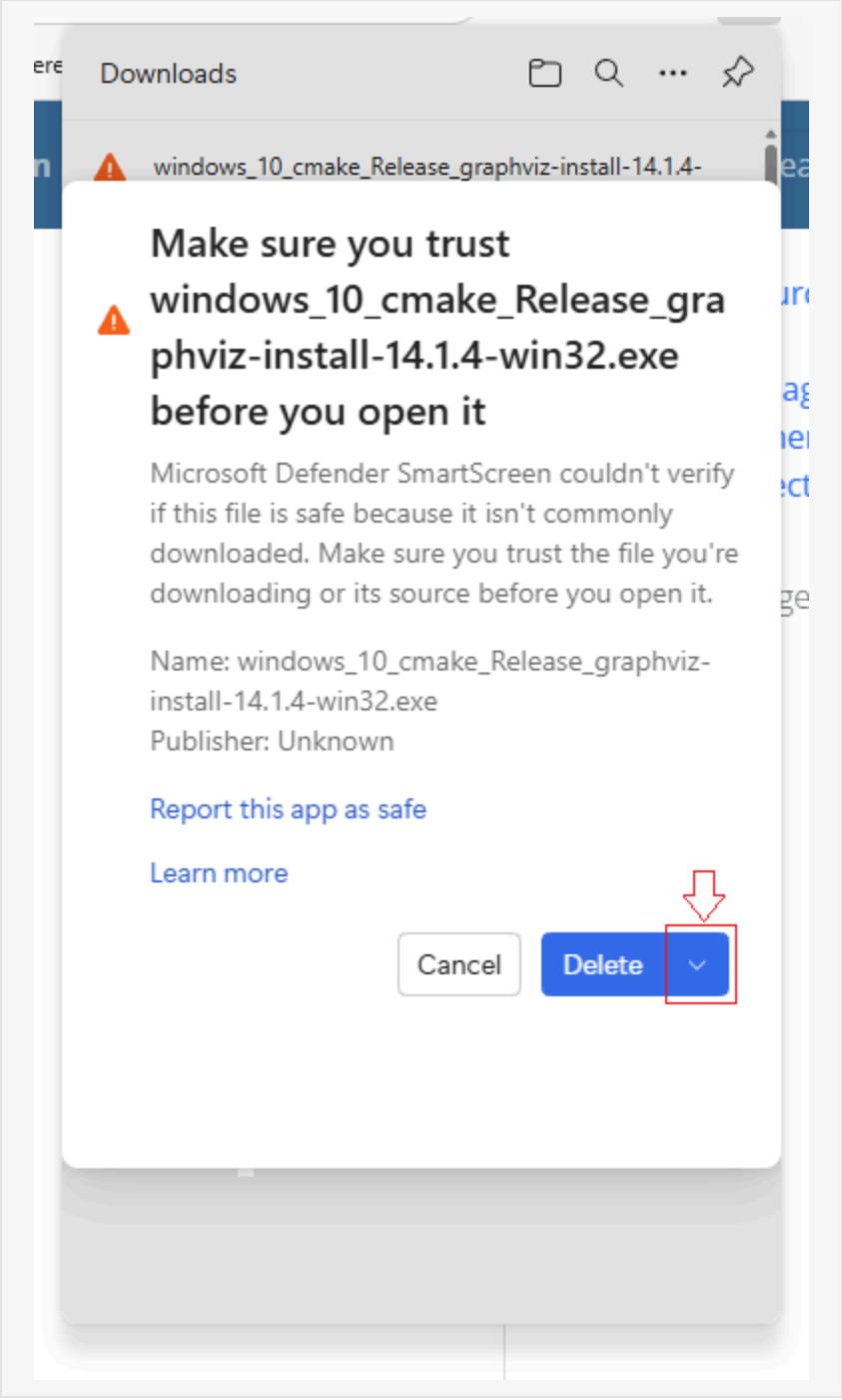


Step 4 - Confirm Again

Microsoft Edge will again try to dissuade you from downloading the file with a warning such as "**Make sure you trust windows_10_cmake_Release_graphviz-install-14.1.4-win32.exe**

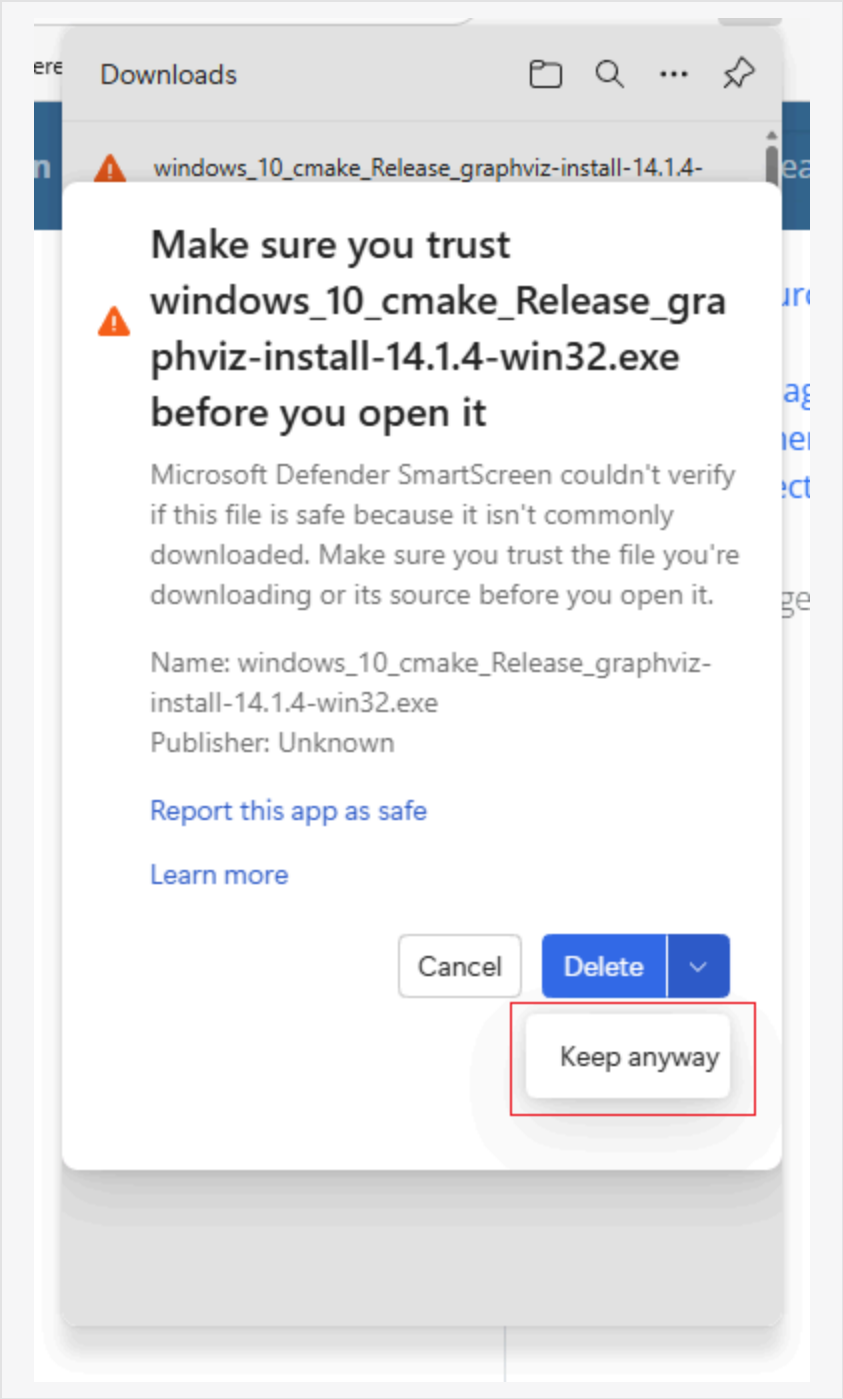
before you open it".

It will appear that your only choices are `Cancel` or `Delete`. To keep the file you must click the `v` dropdown within the Delete button.



An additional choice appears:

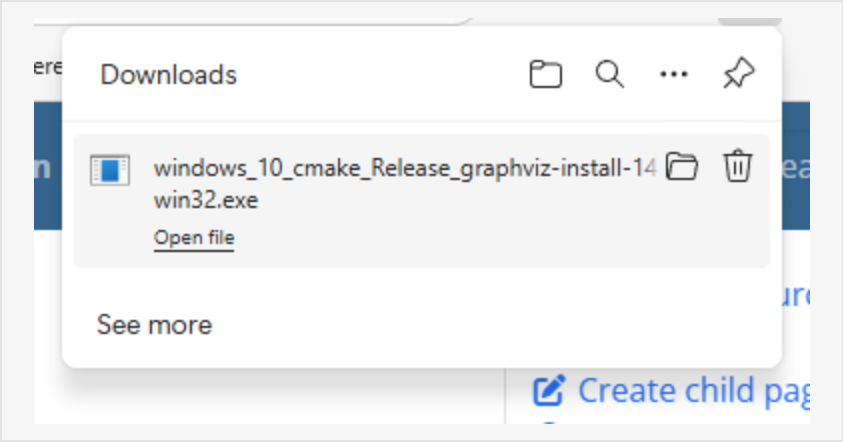
- `Keep anyway`



Click on `Keep anyway`

Step 5 - Open the File

Microsoft Edge downloads the file, and the installer file shows up as a download. Click on `Open file` to run the installer.



Step 6 - Resume Installation

Resume the installation steps

Workbook

The **Relationship Visualizer** is a macro-enabled Excel workbook that lets you collect data using Excel's familiar table format while taking advantage of its full range of functions.

The workbook is organized into multiple worksheets, each serving a specific purpose. Some provide the core functionality, while others support advanced visualizations or presentation-level customization.

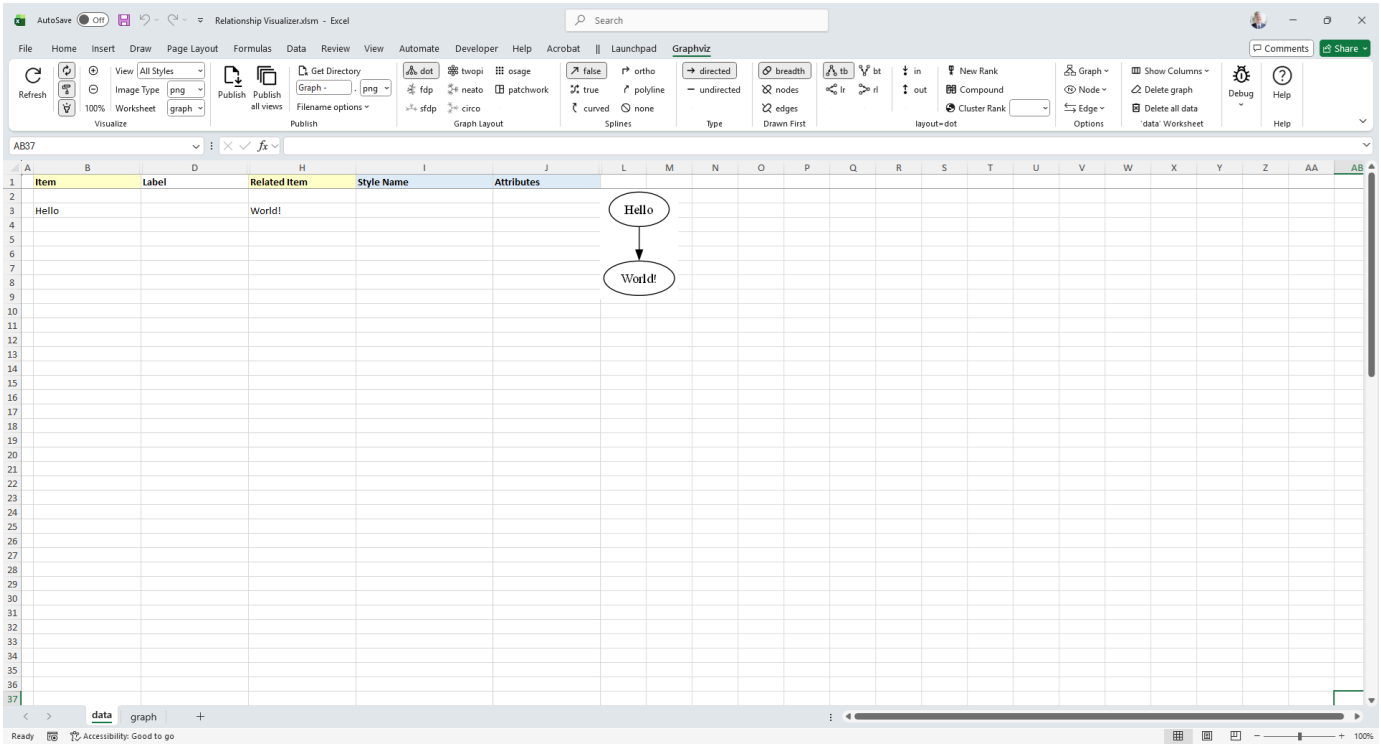
The sections that follow offer a high-level overview of these worksheets to help you navigate the workbook.

`data` Worksheet

The `data` worksheet is the core of the Relationship Visualizer.

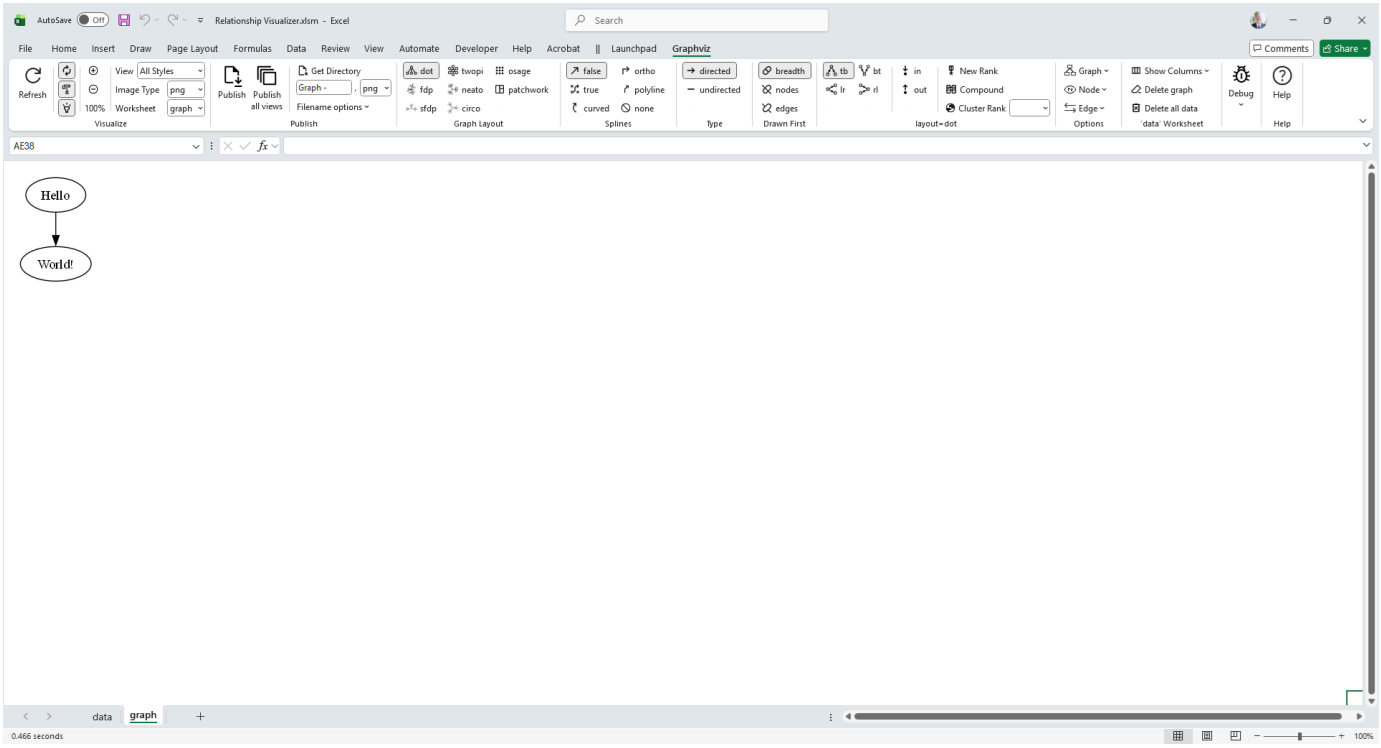
It is where you list the nodes and edge relationships that form the basis of your visualizations.

This worksheet is described in greater detail in Creating Your First Graph.



`graph` Worksheet

The `graph` worksheet displays the image representation of the data from the `data` worksheet whenever the output worksheet is set to `graph` and the **Refresh** button is pressed. This sheet also provides zoom-in, zoom-out, and scrolling capabilities for navigating large graphs.



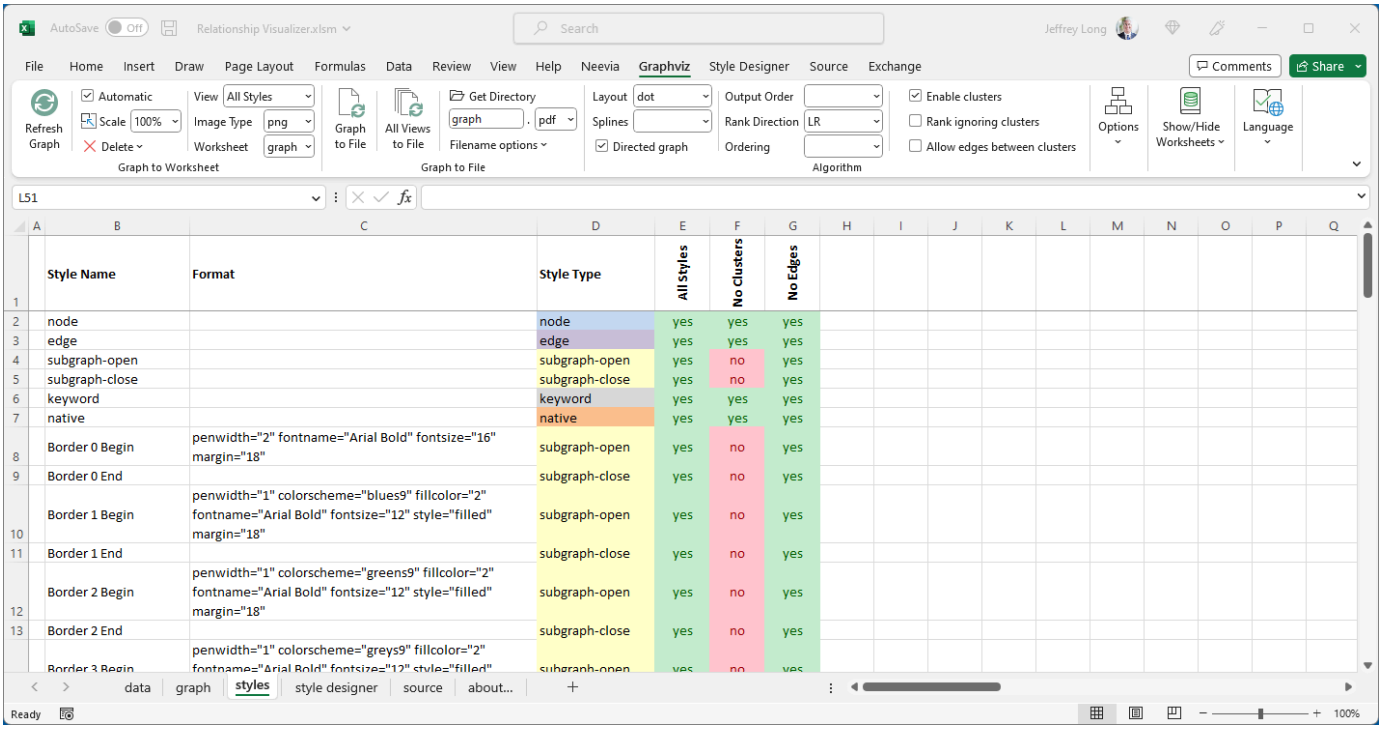
`styles` Worksheet

The `styles` worksheet is where you create style definitions for nodes and edges.

It functions much like an HTML Cascading Style Sheet: you define a style name and specify how that style should appear (shape, color, font, and other visual attributes).

Once defined, a style can be easily associated with multiple nodes or edges in the `data` worksheet.

This worksheet is described in greater detail in Using the `styles` Worksheet.



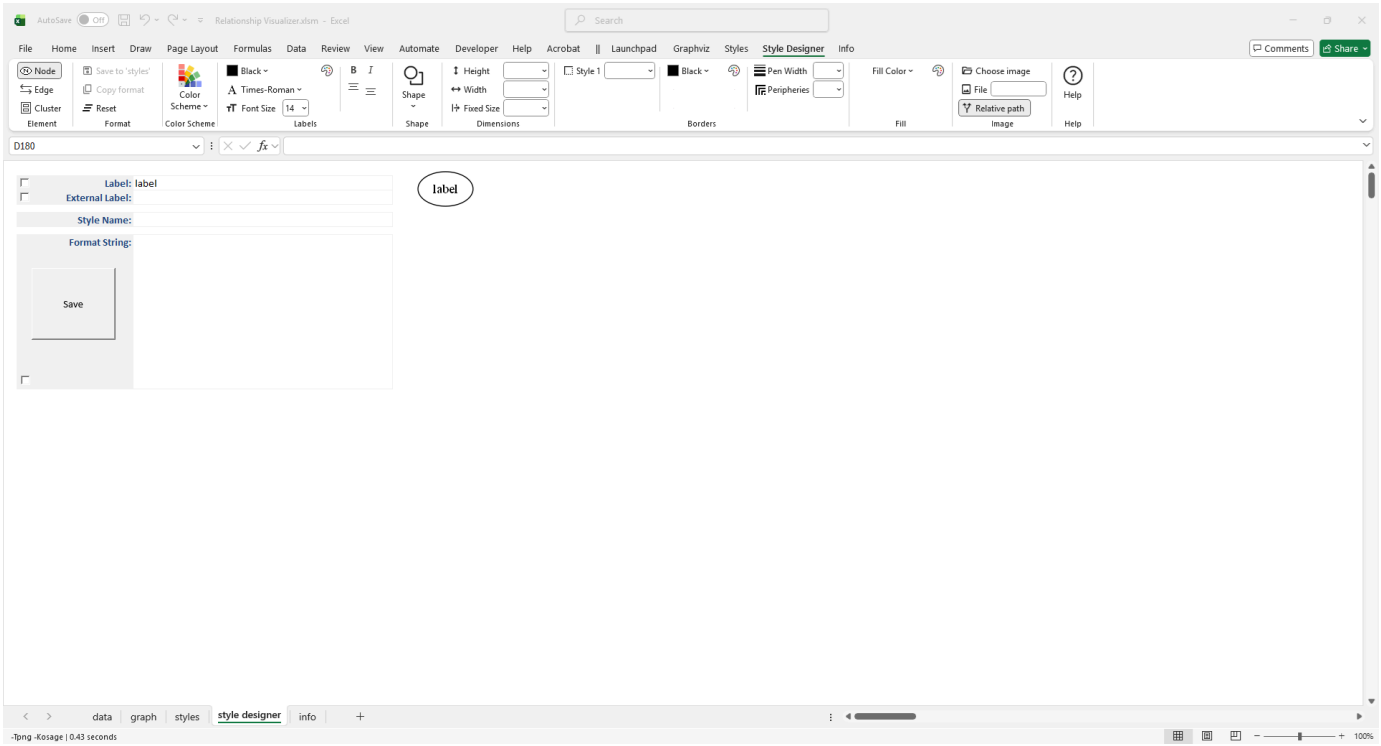
	Style Name	Format	Style Type	All Styles	No Clusters	No Edges
2	node		node	yes	yes	yes
3	edge		edge	yes	yes	yes
4	subgraph-open		subgraph-open	yes	no	yes
5	subgraph-close		subgraph-close	yes	no	yes
6	keyword		keyword	yes	yes	yes
7	native		native	yes	yes	yes
8	Border 0 Begin	penwidth="2" fontname="Arial Bold" fontsize="16" margin="18"	subgraph-open	yes	no	yes
9	Border 0 End		subgraph-close	yes	no	yes
10	Border 1 Begin	penwidth="1" colorscheme="blues9" fillcolor="2" fontname="Arial Bold" fontsize="12" style="filled" margin="18"	subgraph-open	yes	no	yes
11	Border 1 End		subgraph-close	yes	no	yes
12	Border 2 Begin	penwidth="1" colorscheme="greens9" fillcolor="2" fontname="Arial Bold" fontsize="12" style="filled" margin="18"	subgraph-open	yes	no	yes
13	Border 2 End		subgraph-close	yes	no	yes
	Border 3 Begin	penwidth="1" colorscheme="greys9" fillcolor="2" fontname="Arial Bold" fontsize="12" style="filled"	subgraph-open	yes	no	yes

`style designer` Worksheet

The `style designer` makes it easy to create style definitions for nodes and edges. It removes the need to know the specific Graphviz attributes required to achieve a desired visual effect.

The `style designer` lets you adjust settings through dropdown lists and immediately see a preview of how Graphviz will render the node, edge, or cluster. You can fine-tune the attributes until you are satisfied with the appearance, then save the resulting style definition to the `styles` worksheet.

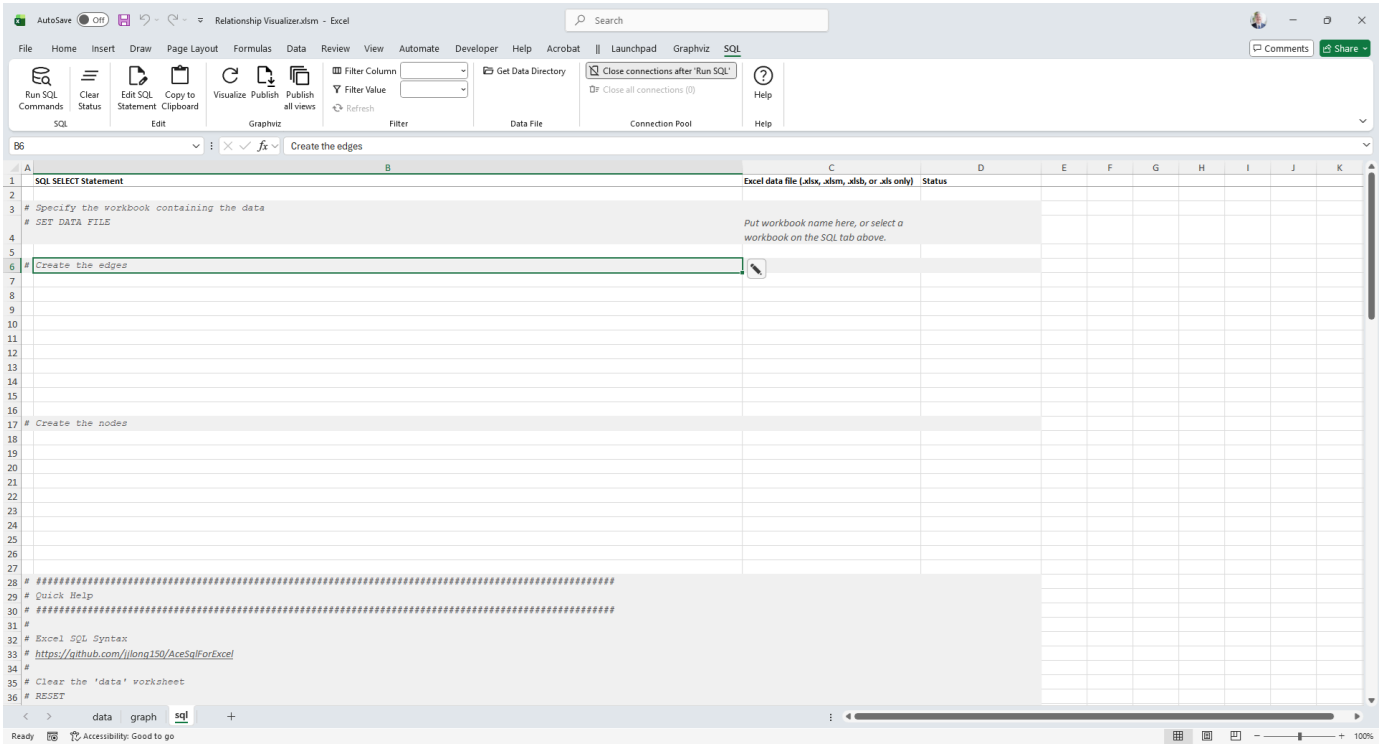
This worksheet is described in more detail in Using the `style designer` Worksheet.



`sql` Worksheet

The `sql` worksheet provides the capability to run SQL statements that pull data from external Excel spreadsheets into the `data` worksheet for graphing.

This worksheet is described in more detail in Using SQL.

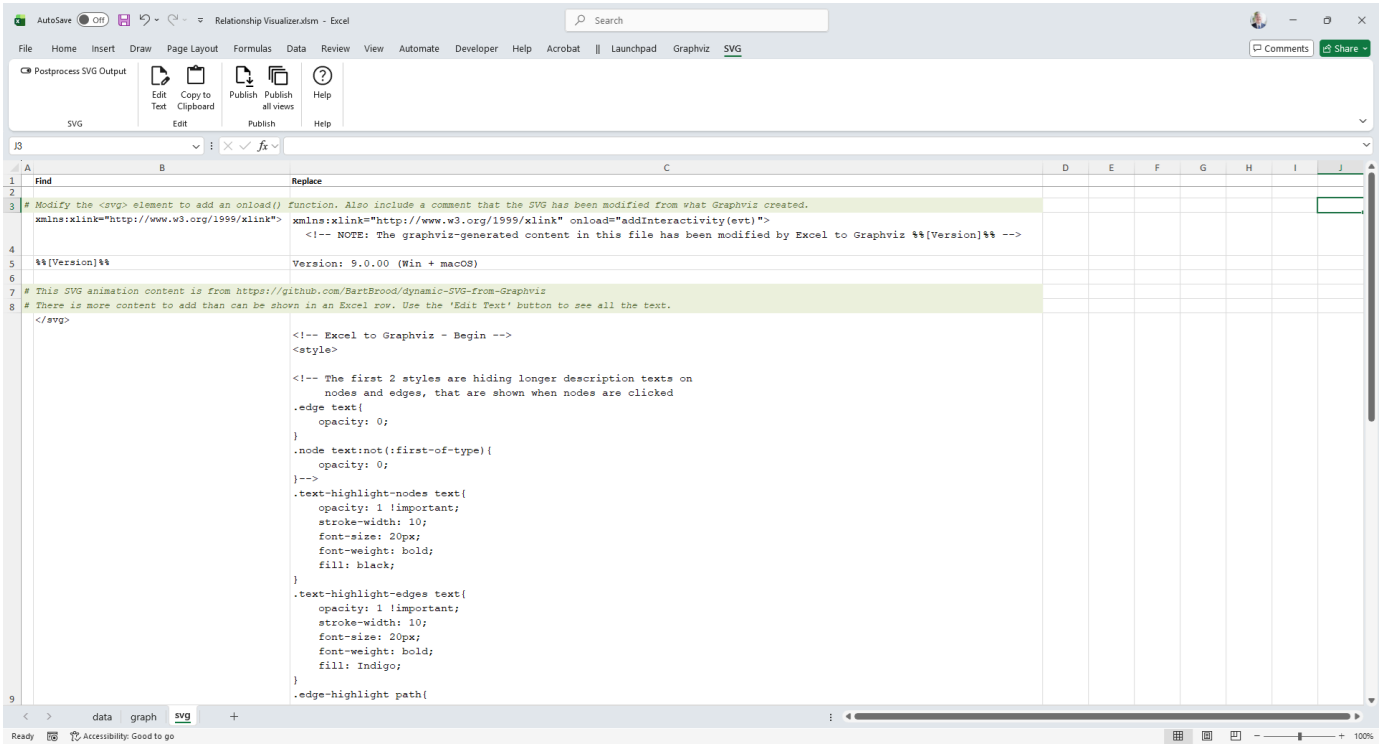


▶ **SQL is not available on Mac OS**

`svg` Worksheet

The `svg` worksheet provides tools for performing find-and-replace operations on graphs published in Scalable Vector Graphics (SVG) format. These operations act as post-processing commands, allowing you to modify the generated SVG—for example, by inserting JavaScript to make the graphs interactive or dynamic.

This worksheet is described in more detail in Post-processing SVG Files.

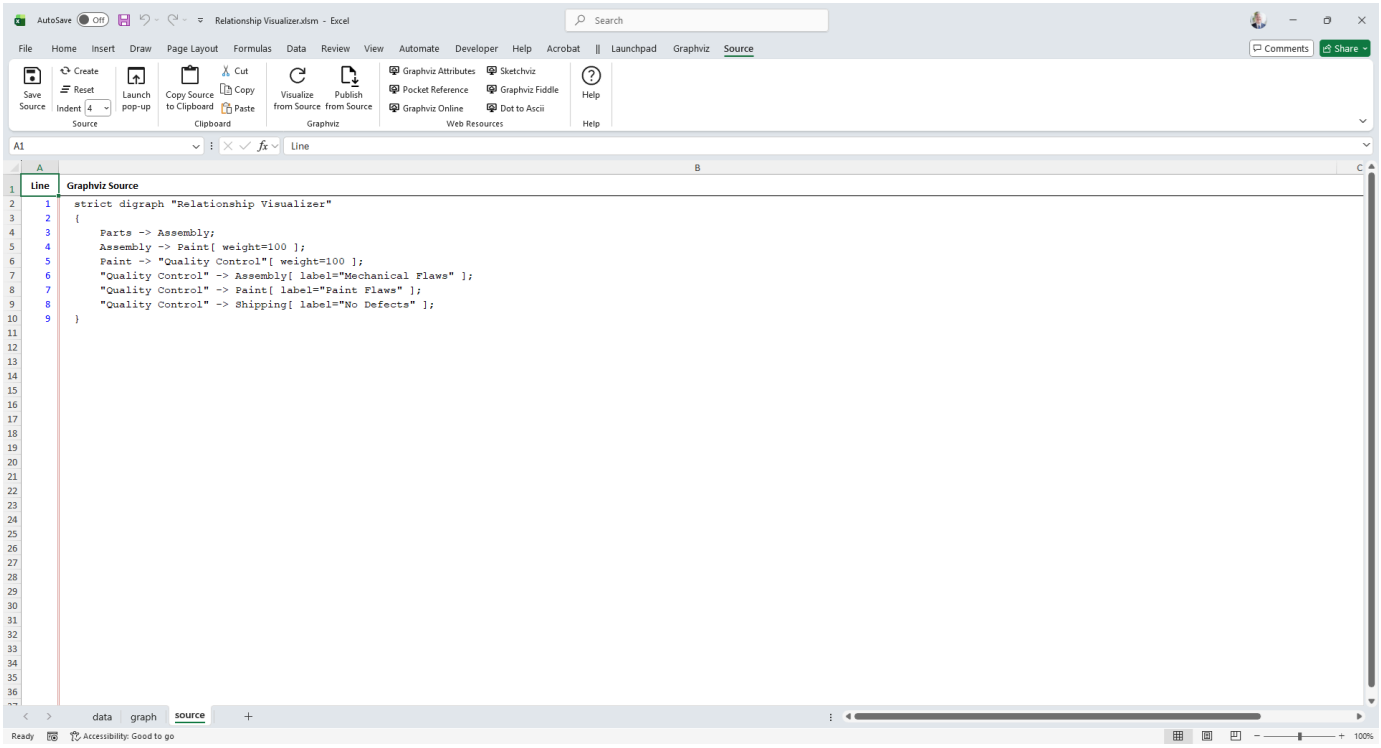


`source` Worksheet

The `source` worksheet displays the DOT language source code generated from the data in the `data` worksheet after a graph has been created.

It also provides links to Graphviz editing and rendering tools, allowing you to experiment with the DOT language outside the workbook.

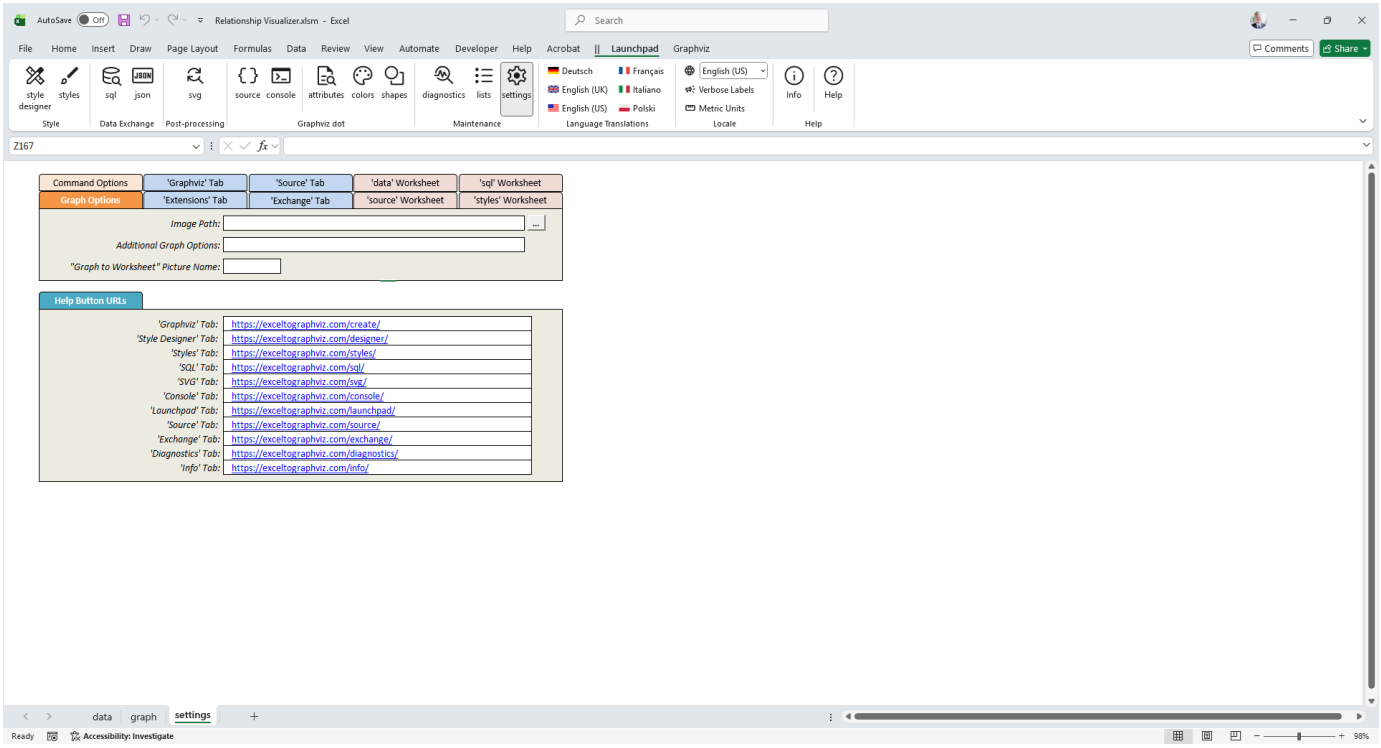
This worksheet is described in more detail in Viewing DOT Source Code.



`settings` Worksheet

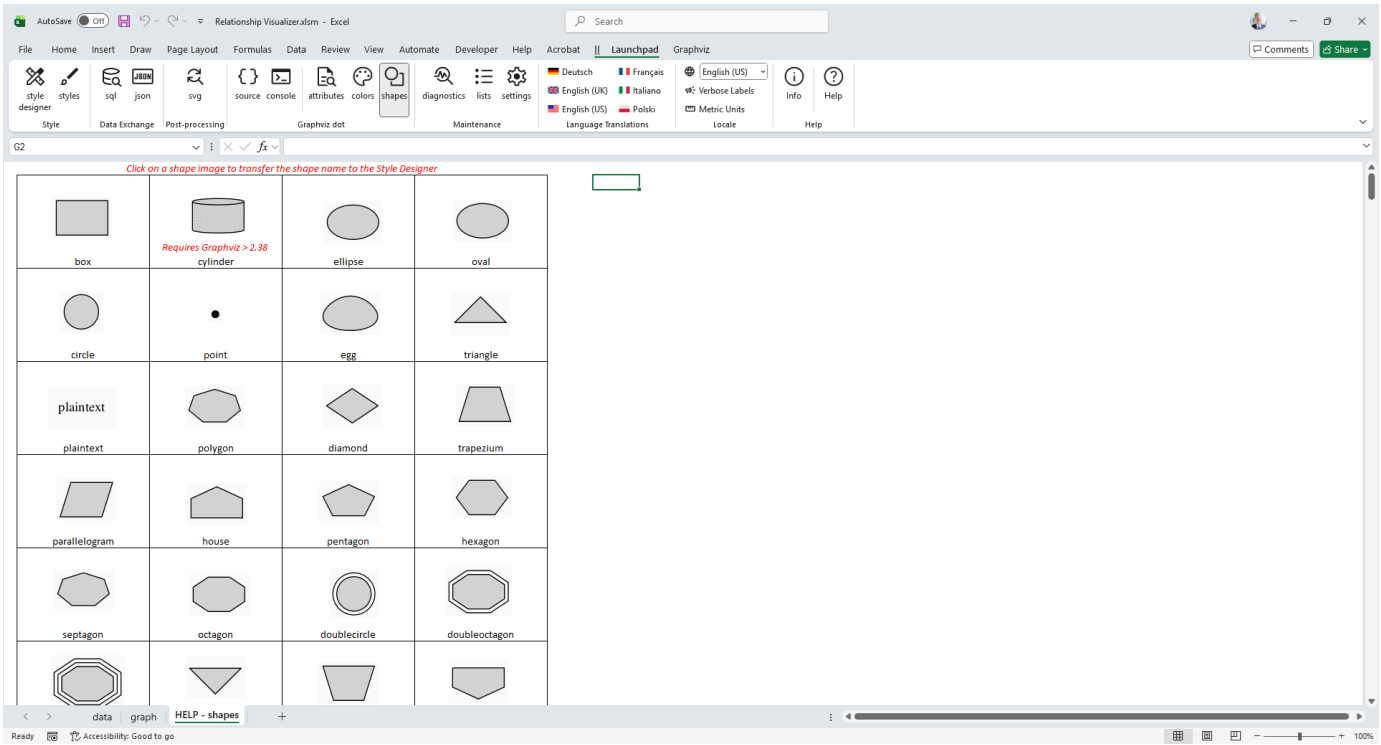
The `settings` worksheet provides options for customizing how Graphviz runs, storing ribbon settings persistently, and configuring behavior across various worksheets.

This worksheet is described in more detail in Changing Master Settings.



`HELP - shapes` Worksheet

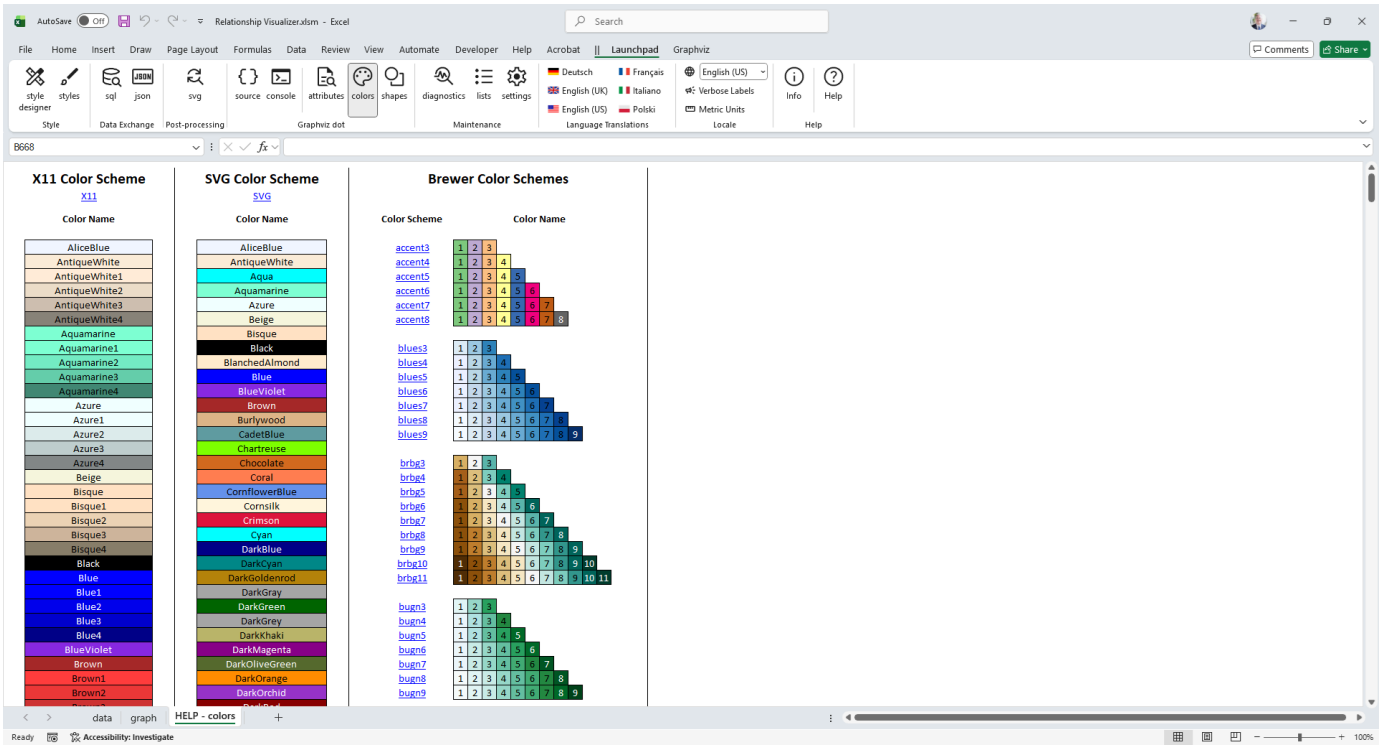
The `HELP - shapes` worksheet provides a glossary of the node shapes supported by Graphviz, along with their corresponding shape names.



`HELP - colors` Worksheet

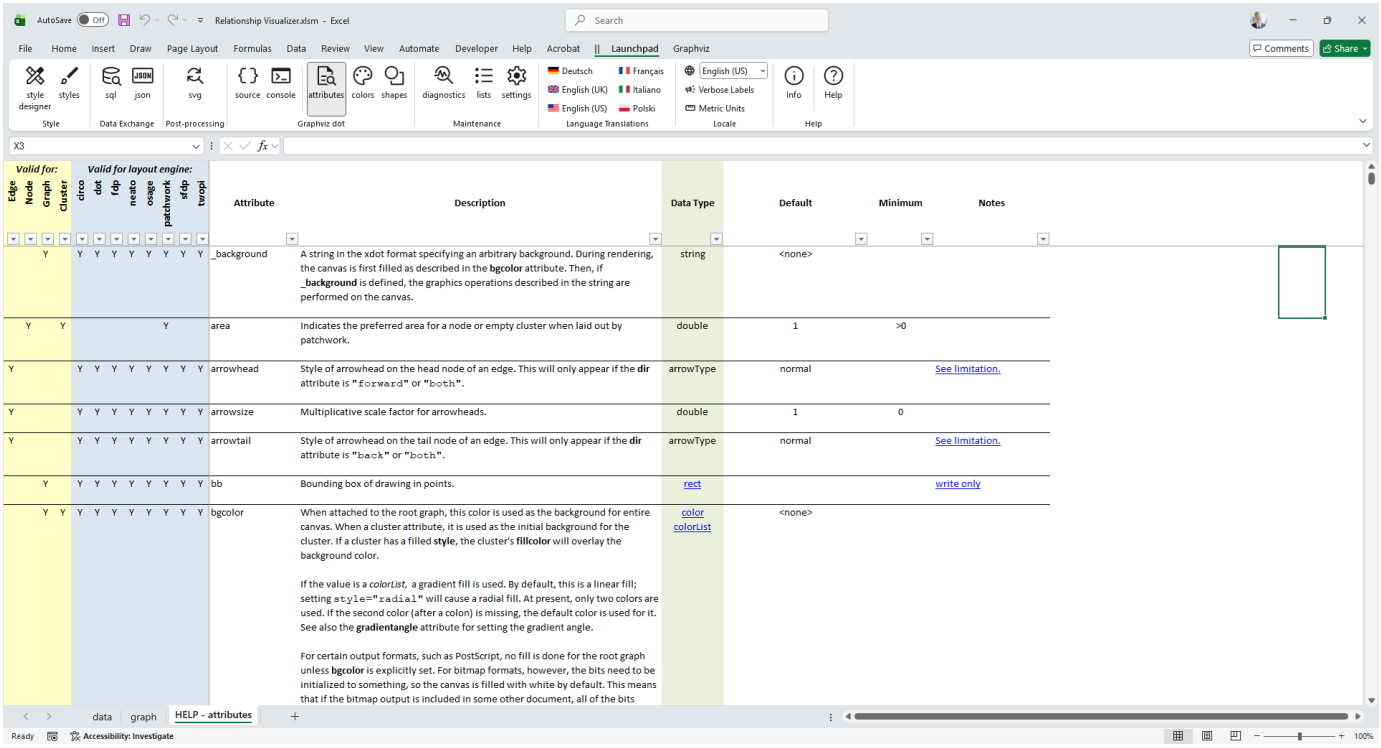
The `HELP - colors` worksheet provides a glossary of the color schemes and color names used by Graphviz, along with a visual sample of each color.

This worksheet is also used to generate the preview images shown in the color dropdown lists on the `style designer` tab.



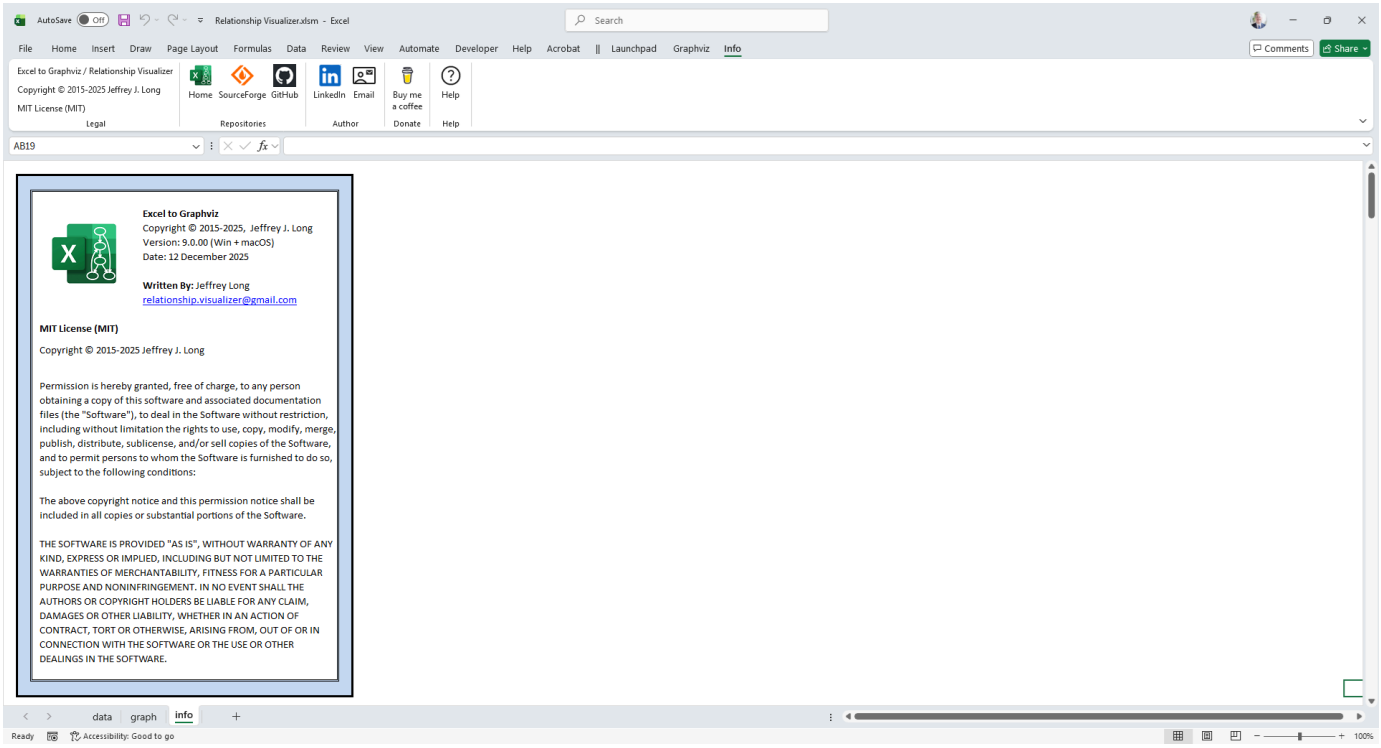
`HELP - attributes` Worksheet

The `HELP - attributes` worksheet provides detailed descriptions of Graphviz language attributes, along with a cross-reference showing where each attribute is used and which Graphviz layout engines recognize it.



`info` Worksheet

The `info` worksheet provides the Relationship Visualizer version number, contact information for the program's author, and the licenses for both the Relationship Visualizer and the open-source components it incorporates.



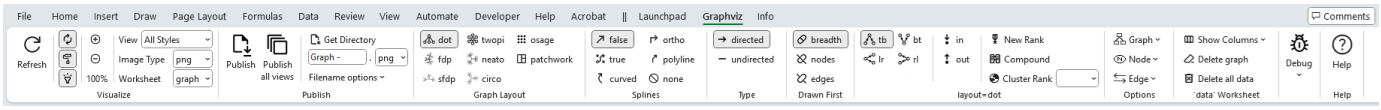
Fluent UI Ribbon Tabs

The Office Fluent ribbon replaced Microsoft's earlier system of layered menus, toolbars, and task panes. The ribbon provides a simpler, more discoverable interface designed to improve efficiency. It includes enhanced context menus, screen tips, a mini toolbar, and keyboard shortcuts that streamline user productivity.

The Relationship Visualizer extends Excel's ribbon interface with additional tabs dedicated to Relationship Visualizer actions, making its features easier to find and use.

`Graphviz` Tab

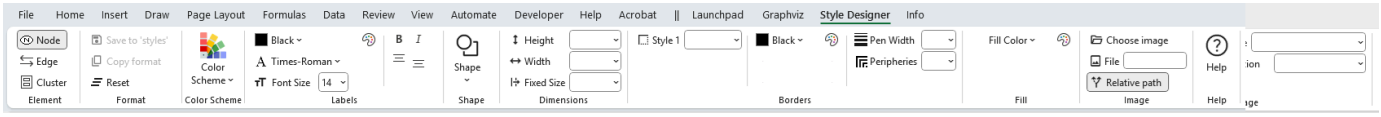
The `Graphviz` tab provides action buttons for creating graphs, along with numerous Graphviz options that control how the graph will be rendered. See The `Graphviz` Ribbon Tab for full details.



`Style Designer` Tab

The `Style Designer` ribbon tab contains the action buttons and settings used to create style-attribute strings for nodes, edges, and clusters. The `style designer` worksheet uses this tab exclusively.

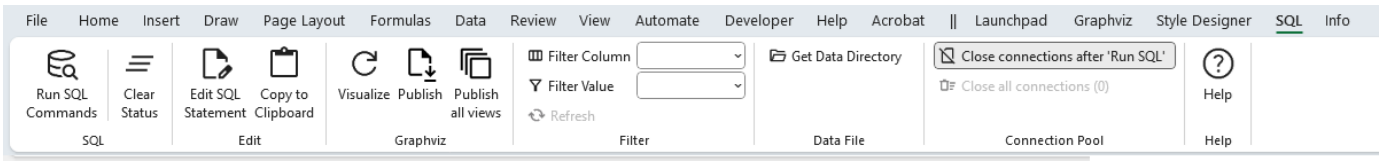
The `Style Designer` ribbon tab is explained in more detail in Using the `style designer` Worksheet.



`SQL` Tab

The `SQL` tab provides action buttons and run-time options for working with the `sql` worksheet. It contains the controls used to run Excel SQL statements, and the `sql` worksheet uses this tab exclusively.

See the section SQL Ribbon Controls for full details.

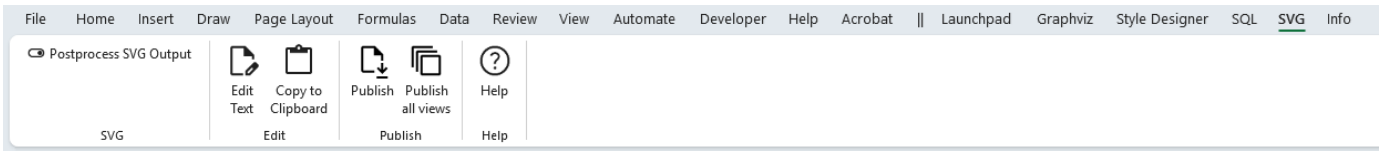


`SVG` Tab

The `SVG` tab provides action buttons and run-time options for working with the `svg` worksheet. It includes a checkbox that enables or disables post-processing of SVG files.

The `svg` worksheet uses this tab exclusively.

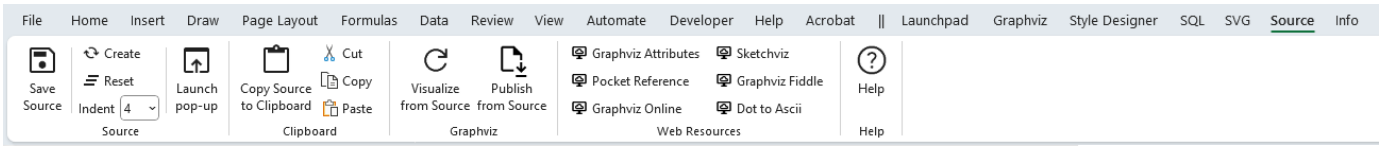
See the section [The `SVG` Ribbon Tab] for full details.



`Source` Tab

The `Source` tab provides action buttons and run-time options for working with the `source` worksheet. It includes controls for creating, viewing, and saving Graphviz source code generated from the information in the `data` worksheet. The `source` worksheet uses this tab exclusively.

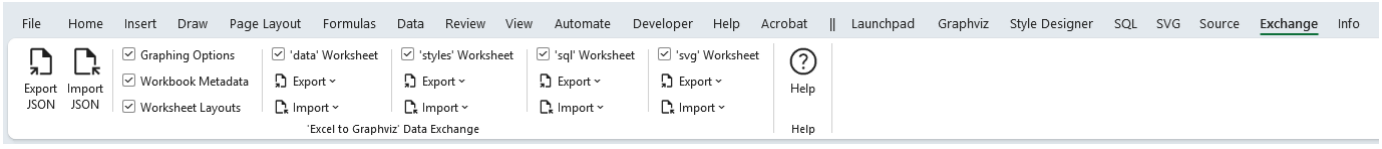
The `Source` ribbon tab is explained in more detail in [Viewing DOT Source Code]. See [The `Source` Ribbon Tab] for full details.



`Exchange` Tab

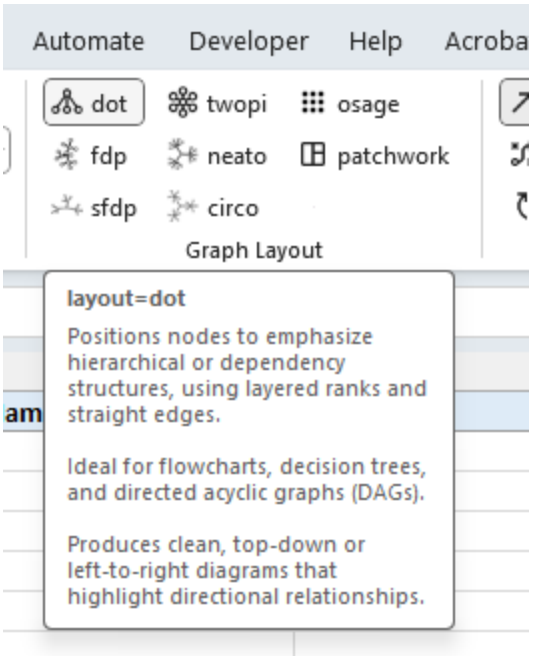
The `Exchange` tab provides action buttons and run-time options for exporting and importing Relationship Visualizer data using JSON text files. This tab is not tied to a specific worksheet, as it works with data drawn from multiple worksheets.

See [The `Exchange` Ribbon Tab] for more details.



Tooltips

All ribbon controls include tool tips that explain their purpose, such as the example shown below for the `dot` layout button on the `Graphviz` ribbon tab. Pause the mouse pointer over any control to display its tool tip.



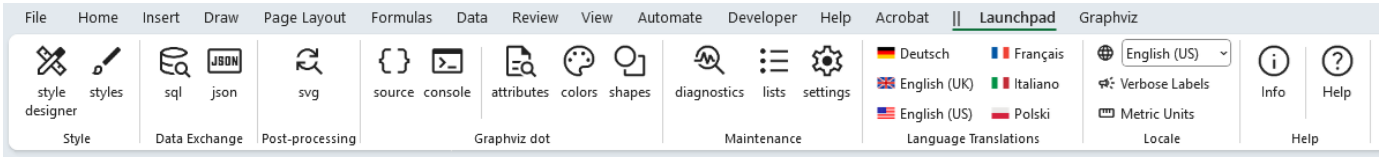
Launchpad

The **Launchpad** provides a convenient way to manage worksheet visibility, allowing you to focus on a small set of worksheets or hide those you rarely use.

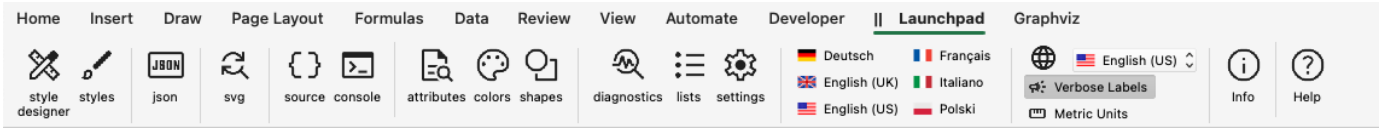
The Launchpad ribbon tab is always present and appears as the first Relationship Visualizer tab after Excel's `Help` tab. The `||` characters in the ribbon serve as a visual divider, marking where Excel's built-in tabs end and the Relationship Visualizer tabs begin.

The tab appears as follows:

`Windows`

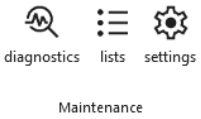
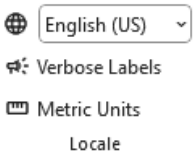


`macOS`



It contains the following groups, each of which is described in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
Style	style designer, styles — Style	Shows the worksheets used to create style definitions, and save them for use in the `data` worksheet.
Data Exchange	sql, json — Data Exchange	Provides access to the tools which can be used to import and/or export workbook data.

Group	Controls	Description
Post-processing		Shows the worksheet which controls the post-processing of `svg` files generated by Graphviz.
Graphviz dot		Provides access to tools which let you see the `dot` source code generated by the Relationship Visualizer, and the messages emitted by Graphviz when the `dot` command is executed to create the graph.
Maintenance		Provides access to the worksheets which are used to maintain the workbook by providing diagnostic information, storing list values used in ribbon tab dropdown lists, or managing run-time settings.
Language Translations		Six language translations are provided.
Locale		Provides an easy-to-find setting which controls which language is displayed in ribbon tabs and worksheet headings, and the units of measure.
Help		Provides the `Help` content for the `Launchpad` ribbon tab.

Style

Shows the worksheets used to create style definitions, and save them for use in the `data` worksheet.

Label	Control Type	Description
style designer	Toggle Button	Show/Hide the style designer worksheet.
styles	Toggle Button	Show/Hide the styles worksheet.

Data Exchange

Provides access to the tools which can be used to import and/or export workbook data.

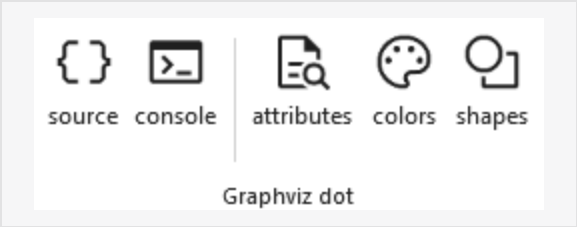
Label	Control Type	Description
sql	Toggle Button	Show/Hide the sql worksheet (Windows only).
json	Toggle Button	Show/Hide the Exchange ribbon tab.

Post-processing

Shows the worksheet which controls the post-processing of `svg` files generated by Graphviz.

Label	Control Type	Description
svg	Toggle Button	Show/Hide the svg worksheet.

Graphviz dot



Provides access to tools which let you see the `dot` source code generated by the Relationship Visualizer, and the messages emitted by Graphviz when the `dot` command is executed to create the graph.

Help worksheets are also provided for those people interested in knowing more details regarding Graphviz.

Label	Control Type	Description
source	Toggle Button	Show/Hide the source worksheet.
console	Toggle Button	Show/Hide the console worksheet.
attributes	Toggle Button	Show/Hide the HELP - attributes worksheet.

Label	Control Type	Description
colors	Toggle Button	Show/Hide the HELP - colors worksheet.
shapes	Toggle Button	Show/Hide the HELP - shapes worksheet.

Maintenance

Provides access to the worksheets which are used to maintain the workbook by providing diagnostic information, storing list values used in ribbon tab dropdown lists, or managing run-time settings.

Label	Control Type	Description
diagnostics	Toggle Button	Show/Hide the diagnostics worksheet.
lists	Toggle Button	Show/Hide the `lists` worksheet.
settings	Toggle Button	Show/Hide the settings worksheet.

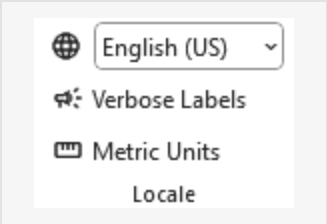
Language Translations

Six language translations are provided. The tool is authored in U.S. English, and those English strings are translated into the other languages using the DeepL Translator ↗.

Because translations may require regional adjustments, access to the translation worksheets was added in Version 7, allowing you to fine-tune the wording to suit your preferences or geographic needs.

Label	Control Type	Description
Deutsch	Toggle Button	Show/Hide the `locale_de-DE` worksheet containing the German translations.
English (UK)	Toggle Button	Show/Hide the `locale_en-GB` worksheet containing the UK English translation.
English (US)	Toggle Button	Show or hide the `locale_en-US` worksheet, which contains the U.S. English translation. This worksheet also provides the default values used whenever a translation is missing from one of the other locale worksheets.
Français	Toggle Button	Show/Hide the `locale_en-FR` worksheet containing the French translations.
Italiano	Toggle Button	Show/Hide the `locale_it-IT` worksheet containing the Italian translations.
Polski	Toggle Button	Show/Hide the `locale_pl-PL` worksheet containing the Polish translations.

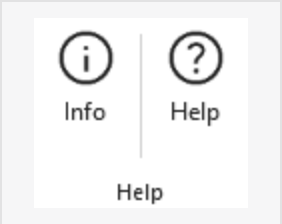
Locale



Provides an easy-to-find setting which controls which language is displayed in ribbon tabs and worksheet headings, and units of measure.

Label	Control Type	Description
Set Language	Dropdown List	Establishes the language translation to use in ribbon tabs and worksheet headings. Changing the language has immediate effect, and you do not need to close and reopen the workbook.
Verbose Labels	Toggle Button	Ribbons on macOS do not display the group names that appear in ribbon tabs on Windows. To compensate, verbose labels were introduced—adding the group name directly into each control's label—so items such as "Color" can be distinguished as "Font Color" or "Fill Color." By default, verbose labels are shown on macOS and short labels on Windows. Users may choose whichever labeling style they prefer.
Metric Units	Toggle Button	Graphviz uses the imperial unit **inches** for size measurements. Metric units were introduced in version 7.0 of the Relationship Visualizer as display units, with automatic conversion to inches performed behind the scenes. Checking this box switches the displayed values from inches to millimeters.

Help

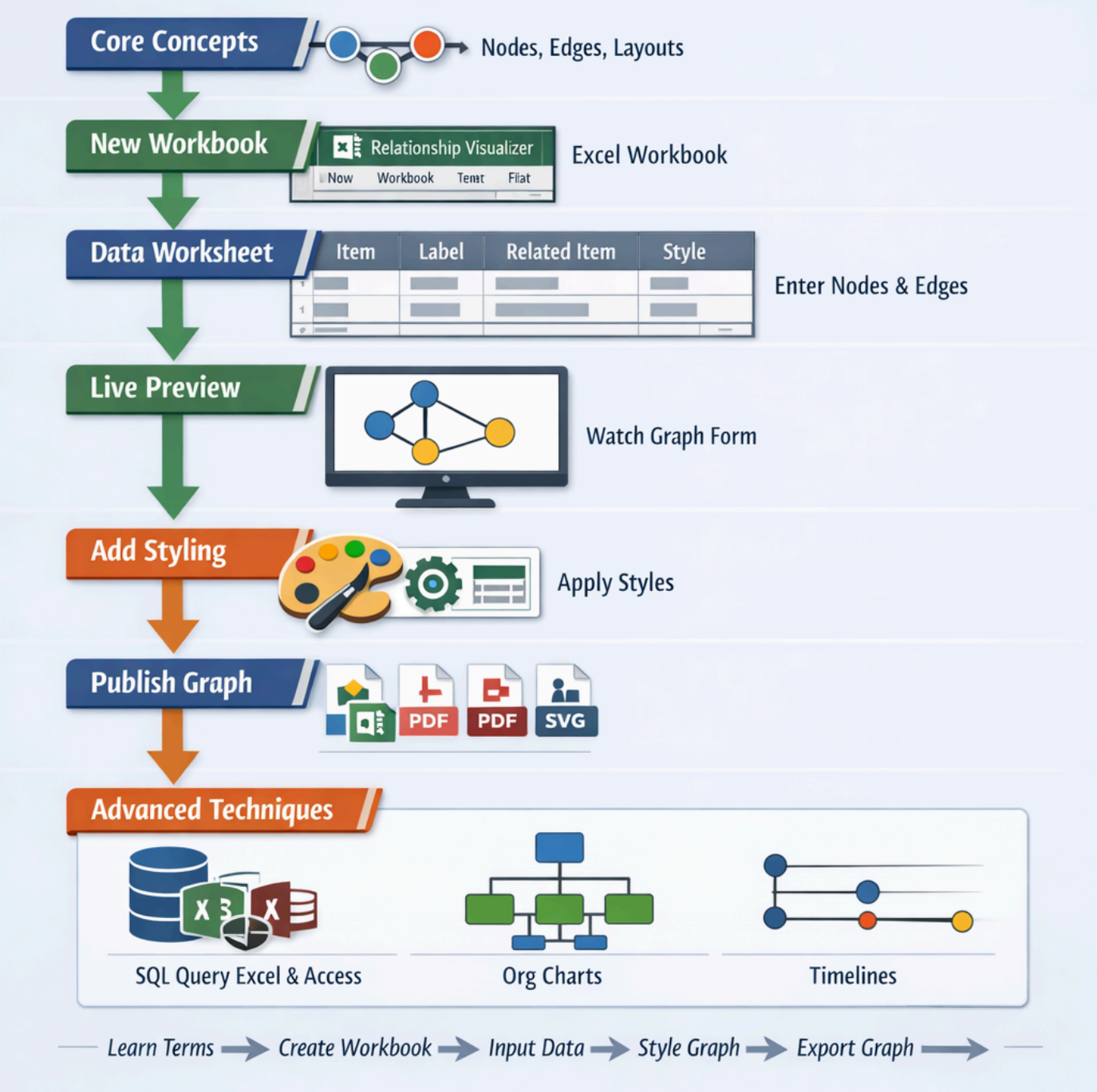


Provides the `Help` content for the `Launchpad` ribbon tab.

Label	Control Type	Description
Info	Toggle Button	Show/Hide the info worksheet.
Help	Button	Provides a link to this web page.

Creating Graphs

Building a graph with the **Relationship Visualizer** spreadsheet is a simple, structured process. You learn the basic vocabulary, create a workbook, enter your data, apply style, and then publish or refine the output.



This page walks through the full workflow and links to focused topics that explain each step in detail.

Learn the Terminology

Before creating your first graph, it helps to understand the basic terms—nodes, edges, labels, styles, and views, as well as the various Graphviz graph layouts. These concepts shape how your data becomes a diagram.

- Terminology

Create a New Workbook

Every graph begins with a `Relationship Visualizer.xlsm` workbook. It provides the worksheets, formulas, and macro automation that turn your spreadsheet entries into Graphviz output.

- Start Here!

Understand the `data` worksheet

The `data` worksheet is where you define the nodes and edges that make up your graph. Each row represents a relationship, and each column adds meaning—labels, types, styles, and more. This sheet is the foundation of every diagram you build.

- `data` Worksheet
- The `Graphviz` Ribbon Tab

Enter Nodes and Edges

Once the workbook and `data` sheet are ready, you can begin entering your node and edge data. As you type, the tool generates a live preview, making it easy to refine structure and relationships as you go.

- Watch the graph form

Add styling

With the structure in place, you shape the visual presentation. Styles let you control color, layout, grouping, and emphasis—helping your graph tell a clearer story.

- Add style to graph elements
- Design reusable styles
- Save styles in a style gallery
- Create style-driven views

Publish your finished graph

When your graph looks the way you want, export it as an image, PDF, or SVG. Formats ideal for documentation, presentations, and sharing.

- Publish Graphs

Enhance your SVG output

If you choose SVG, you can add animation, interactivity, or additional styling using SVG post-processing tools.

- Post-process SVG Files

Explore advanced capabilities

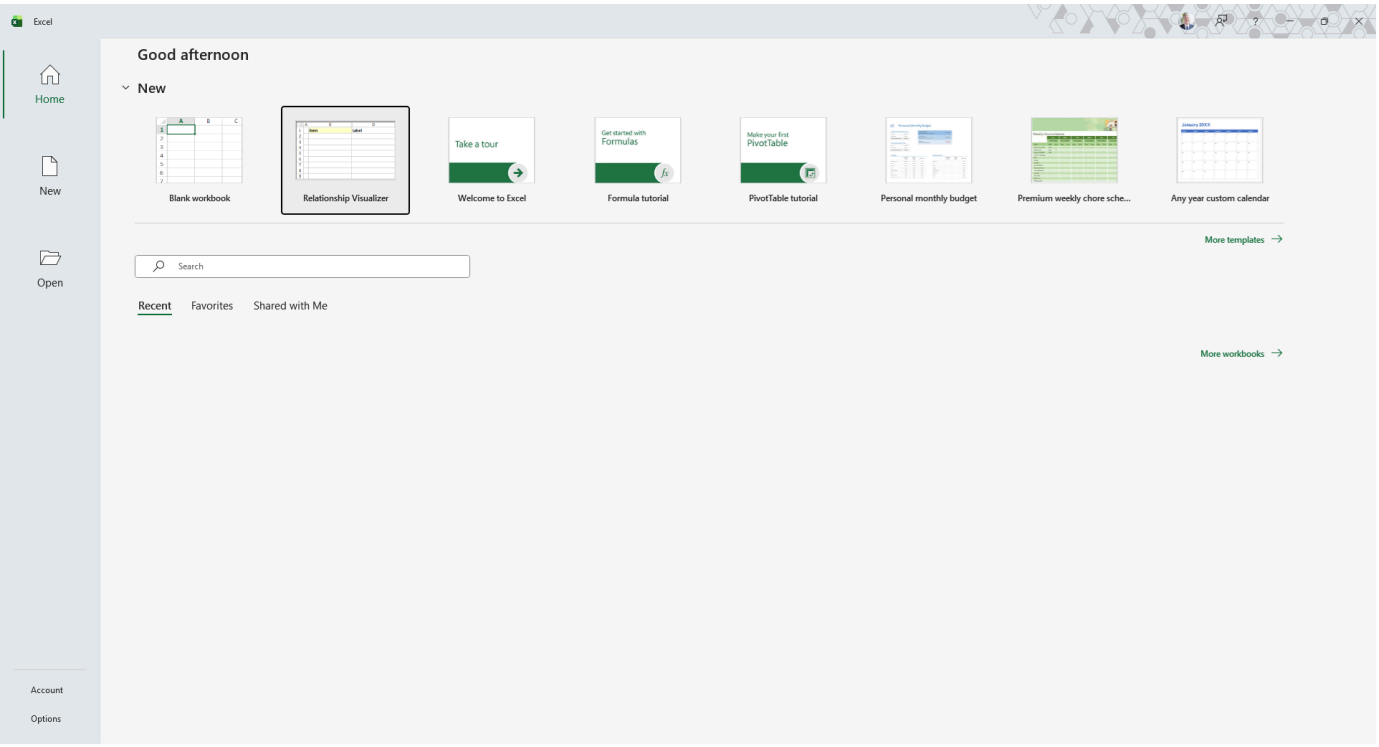
When you're ready to go further, explore advanced features like SQL-driven graph generation, JSON exchange, DOT source inspection, CLI output capture, and expanded configuration options.

- SQL-driven Graph Generation
- JSON Data Exchange
- View Graphviz DOT Source
- Capture Graphviz CLI Messages
- Configure Spreadsheet Settings
- Advanced Graphviz Capabilities

Create a New Workbook

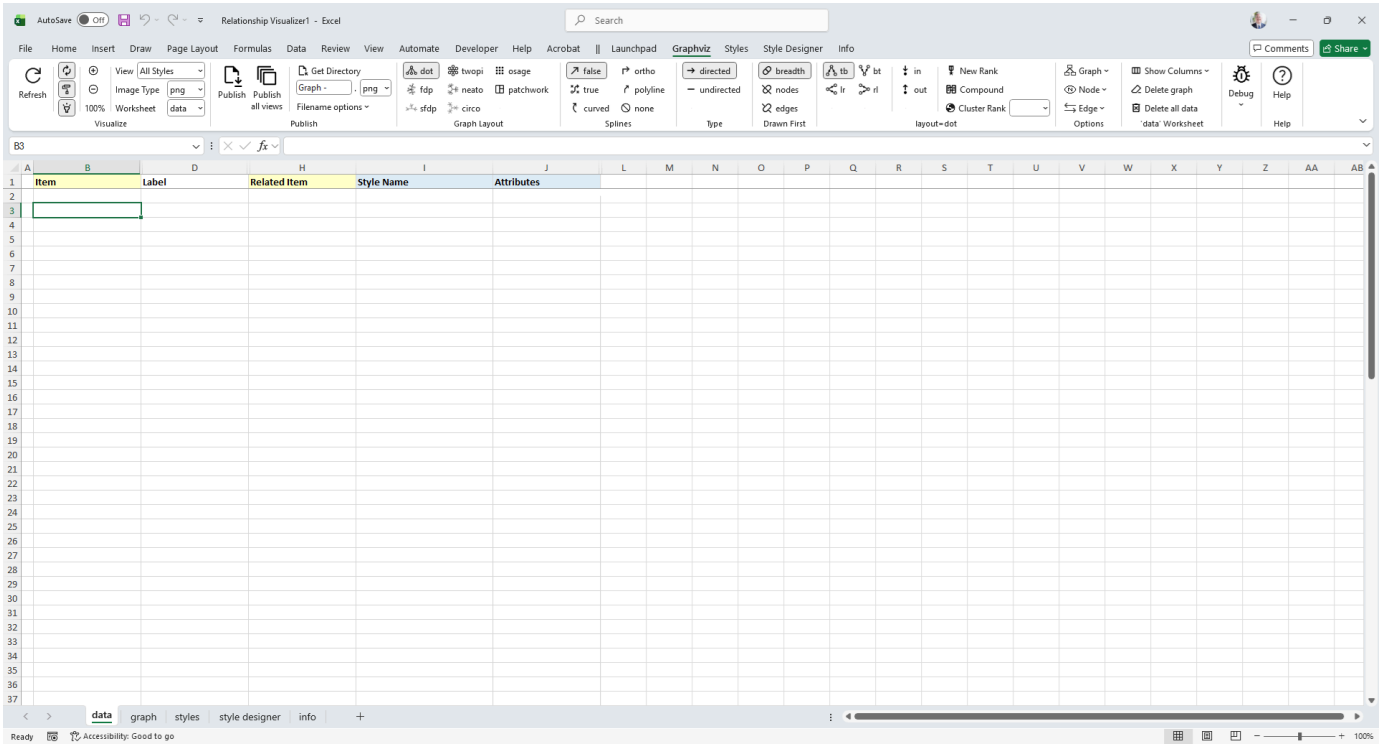
The first action is to launch Microsoft Excel. When Excel starts, it will suggest sample spreadsheets you can create. This will contain the `Relationship Visualizer` template which you saved as a template as part of the installation steps.

Select this template to create a new workbook.



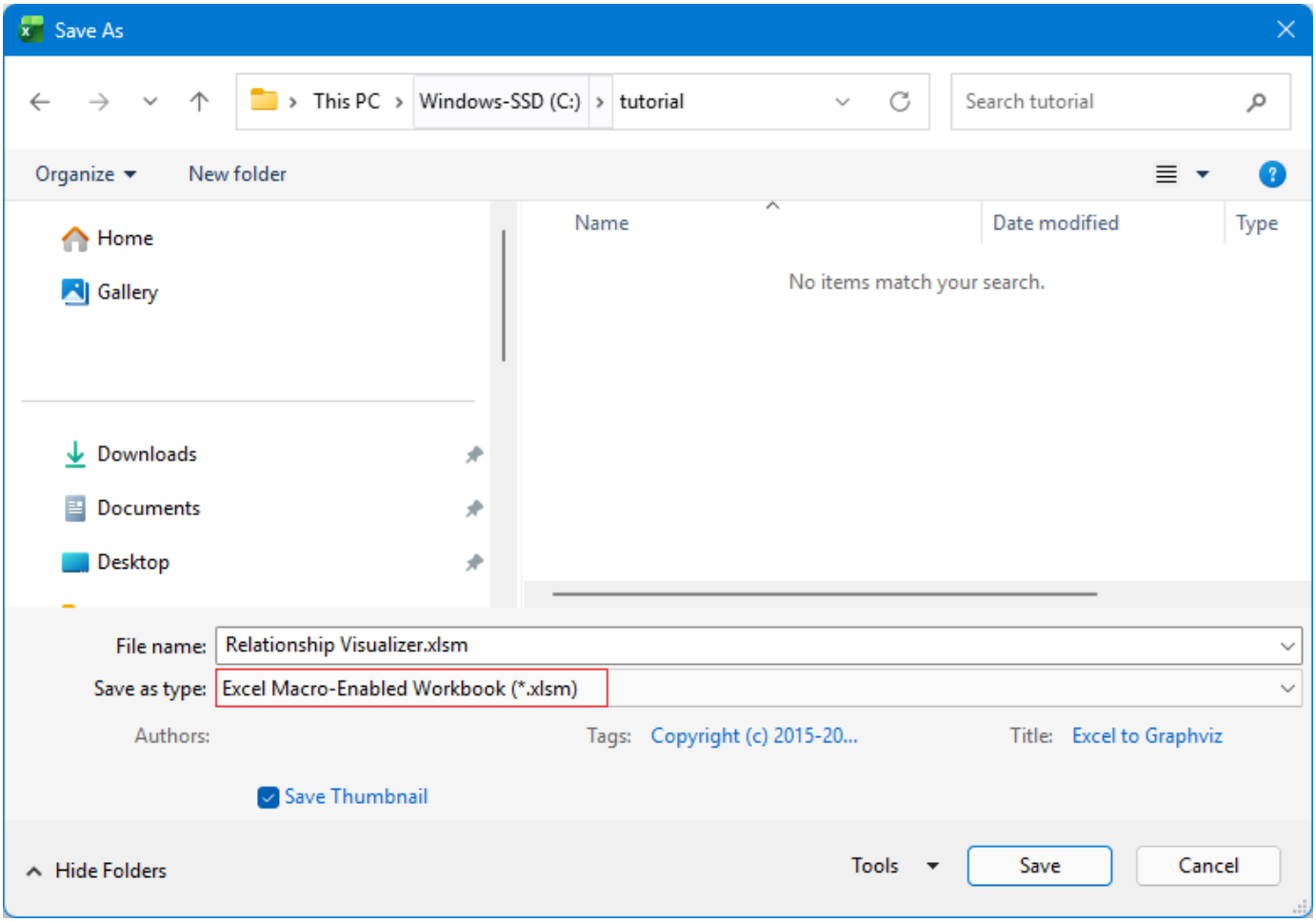
Save the Workbook as a Macro-Enabled Workbook

The workbook will appear as shown below.

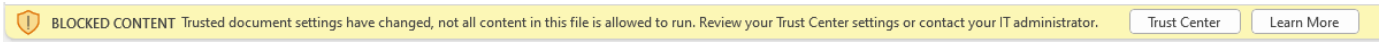


Perform a "**FILE → Save As**" action. Choose a directory where you would like to save the file and change the file name from `Relationship Visualizer1` to something meaningful to you.

The most important step is to set the `Save as type:` dropdown list item as **Excel Macro-Enabled Workbook (*.xlsm)**. You will not be able to run the macros that create the visualizations unless the workbook is *macro-enabled*.

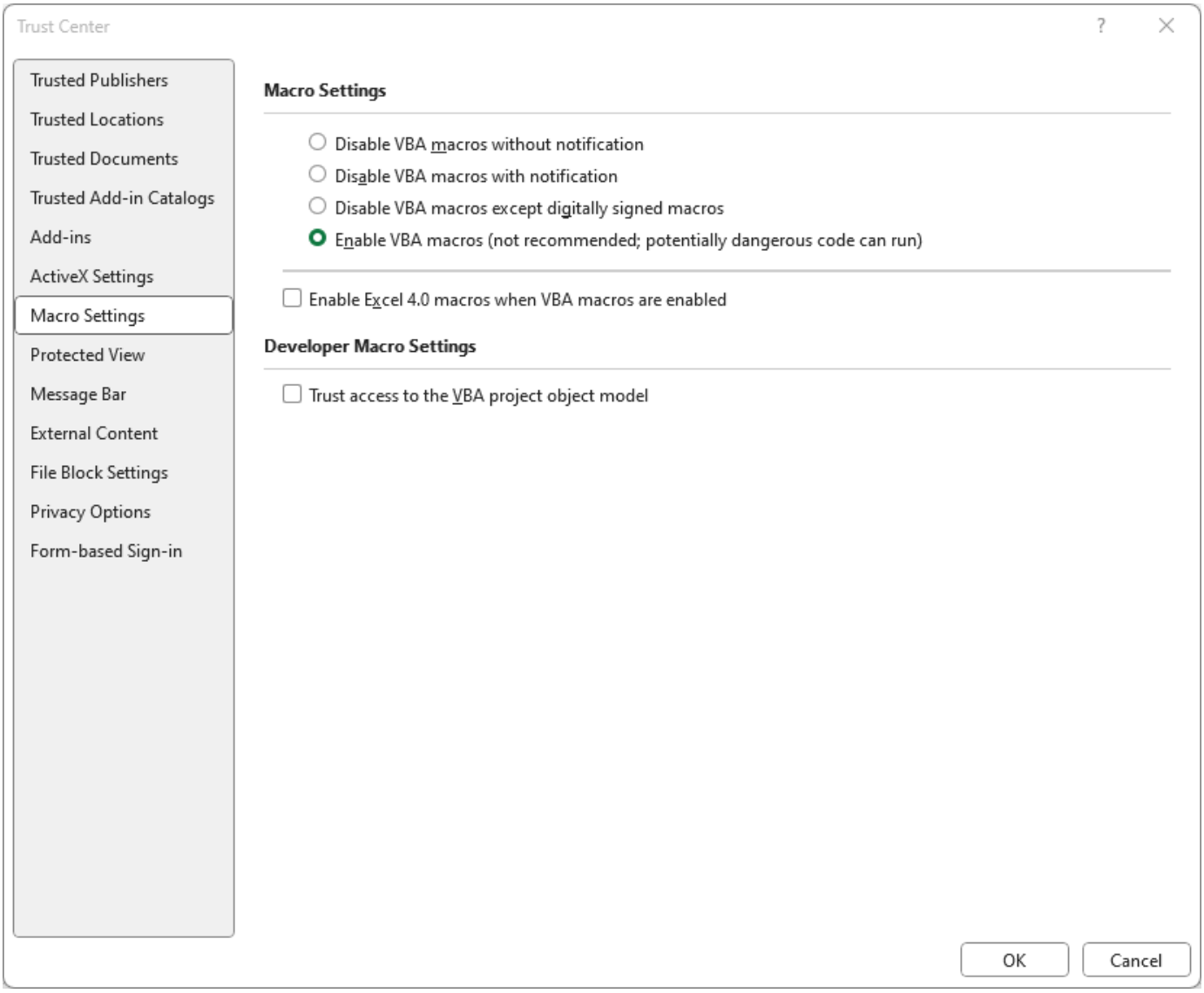


The workbook you just saved may show a **BLOCKED CONTENT** message. If so, click the `Trust Center` button.



The security settings for running macros will be displayed.

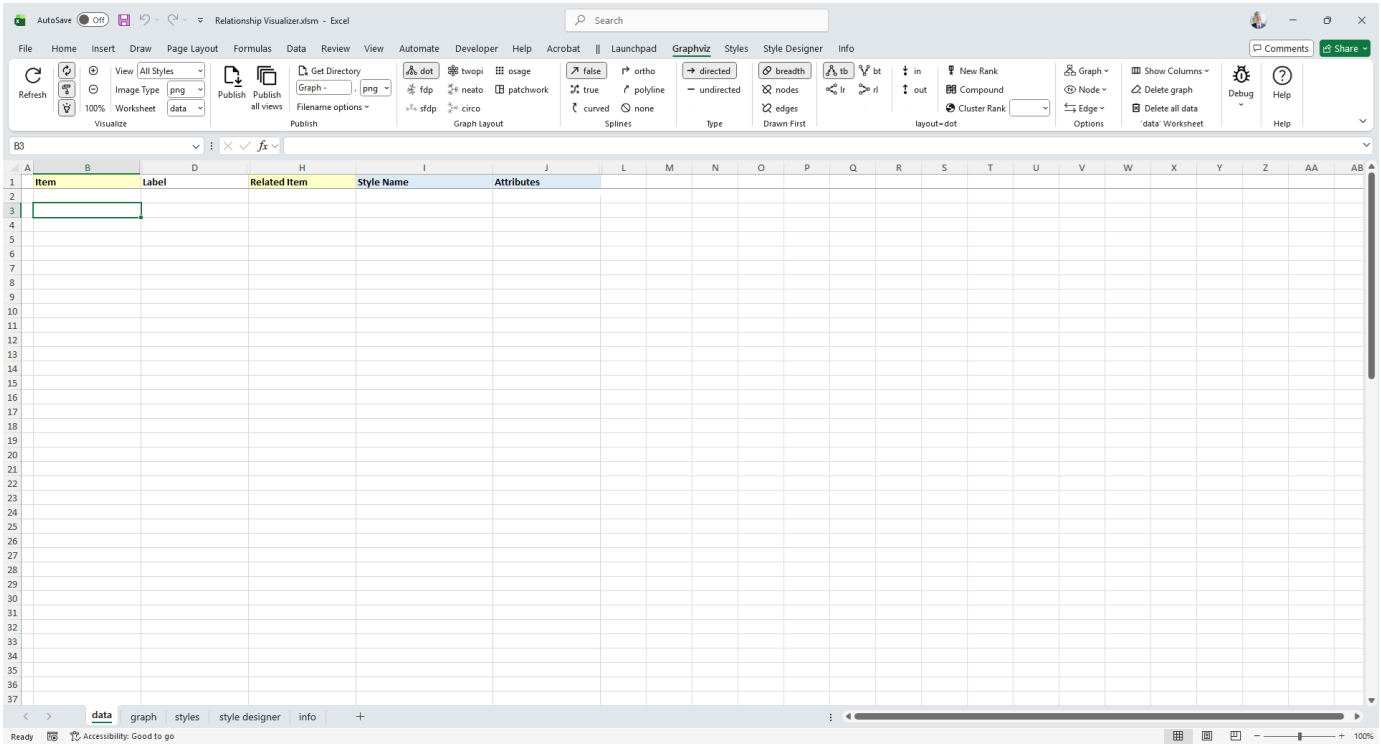
Choose the `Enable VBA macros (not recommended; potentially dangerous code can run)` radio button, and press `OK`.



Close and Reopen the New Workbook

Assuming that you changed the file name from `Relationship Visualizer1` to something meaningful to you, you should now close the file and reopen it.

When you reopen the workbook the message stating that macros have been blocked will be gone. The spreadsheet will appear as follows, displaying a `data` worksheet and a custom ribbon tab named `Graphviz`.



WARNING - Ribbon Fails to Update Dynamically After "Save As"

When you use **File → Save As** to change the workbook's file name, Excel continues to associate the ribbon with the *original* file name. Because of this stale reference, any code that programmatically switches ribbon tabs will stop working.

To work around the issue, you can either manually switch tabs as you move between worksheets, or close and reopen the workbook. Reopening forces Excel to reload the ribbon under the new file name, restoring normal tab-switching behavior.

This is a known issue in **Microsoft Excel** that affects workbooks using a custom ribbon (1 ↗, 2 ↗).

TIP

Any time you save a copy of the spreadsheet using **File → Save As** and change the workbook's file name, you should close the workbook and reopen it. This forces Excel to reload the custom ribbon under the new file name and restores normal tab-switching behavior.

The `data` Worksheet

The `data` worksheet is the core worksheet you will use to create graphs.

Before we create our first graph, lets gain an understanding of the mandatory and optional columns on this worksheet.

The `data` Worksheet has 11 columns (A-K):

A	B	C	D	E	F	G	H	I
Indicator	Item	Tail Label	Label	External Label	Head Label	Tooltip	Related Item	Style Name

---

Indicator

The `Indicator` column is used to draw special attention to a row.

- A `#` hash character treats the row as a comment. The text in the row will turn green, and no data in this row will be included in the graph.
- An `!` exclamation mark character will appear if errors are detected in your data on this row. The row will turn red, and an error message will be displayed in the Messages column.

Item

The `Item` column serves two purposes.

- For nodes, it is a unique identifier of the node.
- For edges, it is the unique identifier of the `from` node in a (`from`, `to`) node pairing.
- **Mandatory** column for nodes and edges.

Tail Label

The `Tail Label` column contains a text label to be placed near the tail of an edge.

- Only used if an edge relationship has been specified.
- Optional column, hidden by default.
- Inclusion in graph can be toggled on/off.

Label

The `Label` column contains text to use to label a node, edge, or cluster.

- When specified for nodes, the value is placed inside the shape.
- When specified for edges, the value is placed near the spline.
- Optional column.
- Inclusion in graph can be toggled on/off.

External Label

The `External Label` column contains text to use to label a node, or an edge.

- When specified for nodes, the value is placed outside the shape, typically above and to the left of the shape.
- When specified for edges, the value is placed away from the spline.
- Optional column, hidden by default.
- Inclusion in graph can be toggled on/off.

If neither a `Label` or `External Label` is specified then the graph will default to showing the `Item` value as the inside label of nodes, and no data for edges.

Head Label

The `Head Label` column contains a text label to be placed near the head of an edge.

- Only used if an edge relationship has been specified.
- Optional column, hidden by default.
- Inclusion in graph can be toggled on/off.

Tooltip

The `Tooltip` column specifies text to be displayed as a tooltip for clusters, nodes, or edges.

- Only applies to graphs saved as files in the `SVG` format.
- Optional column, hidden by default.

Related Item

The `Related Item` column is the unique identifier of the `to` node in a (`from` , `to`) node pairing.

- **Mandatory** column when specifying a relationship (edge).

Style Name

The `Style Name` column indicates which style definition in the `styles` worksheet to use when drawing the graph.

- Optional column.
- Inclusion in graph can be toggled on/off.

Attributes

The `Attributes` column provides a means to add extra elements of style which will only apply to a single row. For example, you can place style attributes in this column to change the color of a key relationship, or the fill color of a key shape you wish to highlight.

- Optional column.
- Inclusion in graph can be toggled on/off.

Messages

When the graphing macros run, they check for common data mistakes, such as specifying only one node for an edge.

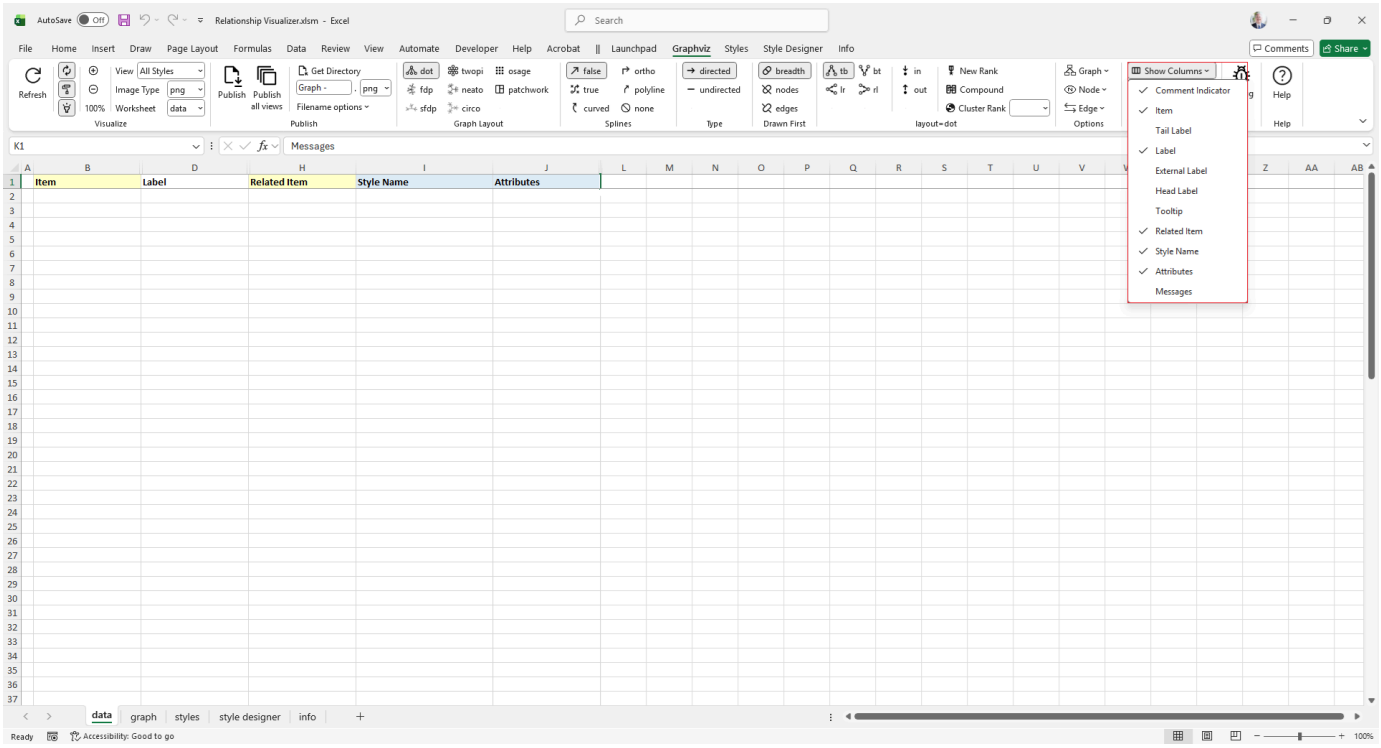
If mistakes are found, they are reported in this column. Additionally, an exclamation mark (!) is placed in the `Indicator` column, and the row is highlighted in red to draw your attention to the error.

- Column is hidden by default.
- Column is shown if an error is detected.

Show Hidden Columns

The columns for `Tail Label` (C), `External Label` (E), `Head Label` (F), `Tooltip` (G), and `Messages` (K) are hidden by default, since they are less frequently used.

You can quickly toggle the visibility of these columns by selecting the column name from the list in the dropdown menu beneath the `Show Columns` button in the **'data' Worksheet** group' on the right side of the Ribbon.

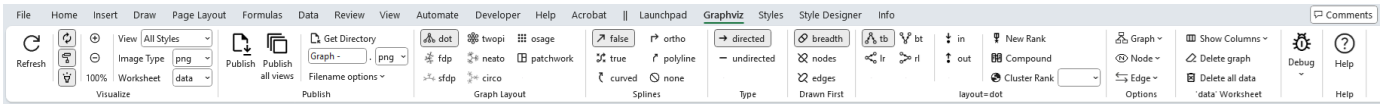

Relationship Visualizer.xlsm - Excel
Search
File Home Insert Draw Page Layout Formulas Data Review View Automate Developer Help Acrobat Launchpad Graphviz Styles Style Designer Info
Comments Share
Refresh
Visualize
Publish
Publish all views
Publish
Graph Layout
Splines
Type
Drawn First
layout=dot
Options
Help
Show Columns
Comment Indicator
Item
Tail Label
Label
External Label
Head Label
Tooltip
Related Item
Style Name
Attributes
Messages
K1
Item
Label
Related Item
Style Name
Attributes
data
graph
styles
style designer
info
Ready
Accessibility: Good to go

The `Graphviz` Ribbon Tab

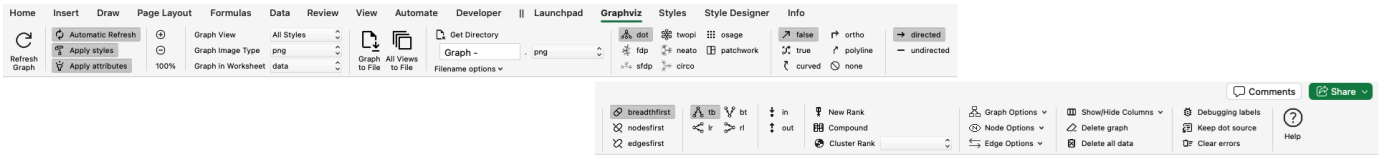
Now that you understand the basics of the `data` worksheet, let's explore the features available in the `Graphviz` ribbon tab.

The `Graphviz` ribbon tab activates automatically whenever any of the following worksheets is selected: `data` `graph`, `styles`, `settings` or `about...`. It looks like this:

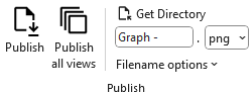
`Windows`

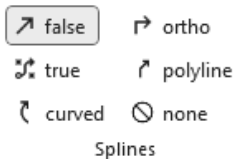
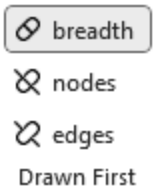
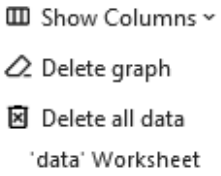
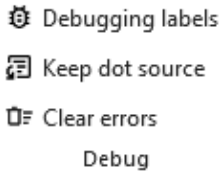


`macOS`

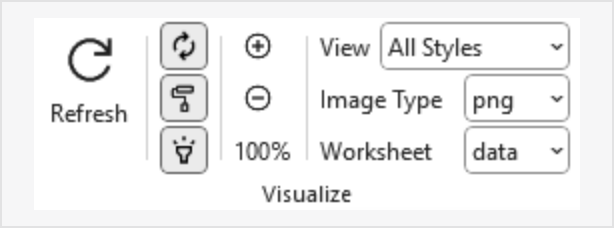


It contains the following groups, which are each explained in content which follows. You may jump directly to the content using the links in this table:

Group	Controls	Description
Visualize		Action and option buttons that cause the Excel data to be graphed by Graphviz and then displayed within the Excel workbook.
Publish		Action buttons that cause the Excel data to be graphed by Graphviz and then written to a file.
Graph Layout		Provides a set of toggle buttons that control which Graphviz layout engine is applied to your diagram.

Group	Controls	Description
Splines		Provides a set of toggle buttons that control how edges are routed in your diagram.
Type		Provides a set of toggle buttons that determine whether your diagram is treated as a directed or undirected graph.
Drawn First		Provides a set of toggle buttons that determine the sequence in which Graphviz draws nodes and edges during rendering.
Layout Options		The Algorithm group within the Graphviz tab changes dynamically based upon the layout algorithm chosen. The graph options shown are specific to that particular layout algorithm.
'data' Worksheet		A set of menu items that control what columns and graphs are displayed on the `data` worksheet.
Debug		An option to display additional information such as the row number and Item identifiers in the labels of nodes, edges, and clusters.
Help		Provides a link to the `Help` content for the `data` worksheet (i.e. this web page).

Visualize

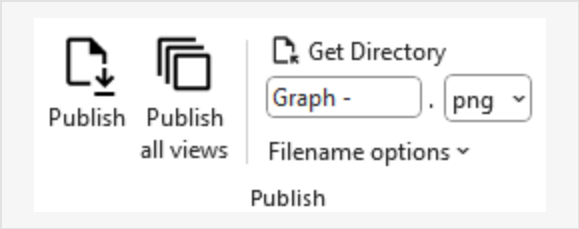


Label	Control Type	Description
Refresh	Button	The action button that causes the Excel data to be graphed by Graphviz and then displayed within the Excel workbook.
Automatic	Toggle Button	When selected, keystrokes are monitored and as cell changes are detected the graph is automatically refreshed (also requires that `Worksheet` is set to `data`).
Apply Styles	Toggle Button	Specifies if the style attributes associated with the Style Name assigned to a node, edge, or cluster should be applied when the graph is generated. **Choices:** • *Pressed* - use the style format • *Unpressed* - do not use the style format (i.e., use default Graphviz rendering method)
Apply Attributes	Toggle Button	Specifies if the style attributes in the `Attributes` column on the `data` worksheet should be included or omitted when the graph is generated. **Choices:** • *Pressed* - include the style attributes • *Unpressed* - do not include the style attributes

Label	Control Type	Description
Zoom In	Button	Magnifies the scale the of image displayed in Excel by 5%.
Zoom Out	Button	Decreases the scale the image displayed in Excel by 5% so as the graph gets larger, you can see more of it within the workbook without having to scroll.
Current Zoom	Text	Shows the current magnification level. The magnification can range from 5% to 150% in 5% increments
View	Dropdown list	The name of the column in the `styles` worksheet which controls which set of Yes/No values to use when creating the diagrams. This dropdown list is explained in more detail in the section Creating Views.
Image Type	Dropdown list	Image format to use when displaying the graph on the `data` or `graph` worksheet of the Relationship Visualizer. **Choices:** • `bmp` - Microsoft Windows Bitmap format • `gif` - Graphics Interchange Format • `jpg` - Joint Photographic Experts Group format • `png` - Portable Network Graphics format • `svg` - Scalable Vector Graphics **Note:** SVG images only display in Office 365; they do not display in older versions of Excel.
Worksheet	Dropdown list	The worksheet in the current workbook where the graph should be displayed **Choices:** • `data` - The graph is displayed in the `data` worksheet to the right of the data columns.

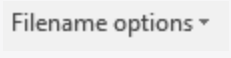
Label	Control Type	Description
		• `graph` - The graph is displayed in the `graph` worksheet, and the `graph` worksheet is activated. This setting is useful for large graphs as it allows you to use Excel's magnification Zoom-In/Zoom-out feature. It is also useful when you want to flip back and forth between the data and the graph to correct errors in the data.

Publish



A tutorial on how to use these ribbon options is contained in the section Publishing Graphs.

Label	Control Type	Description
Publish	Button	The action button that causes the Excel data to be graphed by Graphviz and then written to a file.
Publish all views	Button	The action button that causes the Excel data to be graphed by Graphviz and then written to a file repeatedly for every view defined in the `Styles` worksheet.
Get Directory	Button	Brings up the Directory Selection dialog and stores/displays the directory where the files should be written to. Once a directory is selected the directory path replaces the "Get Directory" button label.
File Prefix	Edit box	Base portion of the file name. For example: `Graph` .

Label	Control Type	Description
		You may also build a file name using the following character strings in the file prefix to insert run-time values into the file name. • `%D` - Current date • `%T` - Current time • `%E` - Graphviz layout engine • `%S` - Splines • `%V` - View name • `%W` - Worksheet name **NOTE**: You must check the appropriate options in the `Filename options` dropdown list for the substitutions to occur.
File Format	Dropdown List	File format of the output file. **Choices:** • `bmp` - Microsoft Windows Bitmap format • `gif` - Graphics Interchange Format • `jpg` - Joint Photographic Experts Group format • `pdf` - Portable Document Format • `png` - Portable Network Graphics format • `ps` - Postscript format • `svg` - Scalable Vector Graphics format • `tiff` - Tagged Image File Format
Filename options 	Dropdown List	A list of options which can be checked which will cause run-time information to be appended or omitted from the file name.

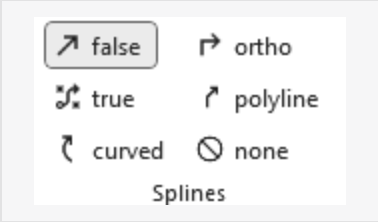
Label	Control Type	Description
Add date/time to the filename	Check box	Option to add a date and time to the file name. **Choices:** • *Checked* - Add the date and time • *Unchecked* - Omit the date and time
Add Layout/Splines to the filename	Check box	Option to add the layout engine and spline type to the file name. **Choices:** • *Checked* - Add the options • *Unchecked* - Omit the options

Graph Layout

The **Graph Layout** section provides a set of toggle buttons that control which Graphviz layout engine is applied to your diagram. These toggles function like radio buttons, ensuring that only one layout is active at a time. This approach gives you a quick, intuitive way to explore how different layout algorithms organize your graph

Splines

The **Splines** section provides a set of toggle buttons that control how edges are routed in your diagram. These toggles function like radio buttons, ensuring that only one spline style is active at a time. This design gives you a quick, intuitive way to explore different routing options—straight lines, curves, orthogonal paths, and more—and immediately see how each style affects the readability and structure of your graph



Button	Description
false	Edges are drawn as straight lines.
true	Edges are drawn using a combination of straight segments and free-flowing curves.
curved	Edges are drawn as smooth, continuous curves between nodes.
ortho	Edges are routed using horizontal and vertical segments with 90-degree bends.
polyline	Edges are drawn as straight segments with angular bends (not restricted to right angles).
none	Edges (and edge labels) are not drawn, but still influence node placement.

Graph Type

The **Graph Type** section provides a set of toggle buttons that determine whether your diagram is treated as a directed or undirected graph. These toggles function like radio buttons, ensuring that only one graph type is active at a time. This setup gives you a quick, intuitive way to switch between directional and non-directional relationships and immediately see how edge arrows, routing, and layout behavior change in response.

Button	Description
undirected	Creates an Undirected Graph graph. Edges have no direction and are drawn without arrowheads.
directed	Creates a Directed Graph graph (digraph). Edges have a defined direction and are drawn with arrowheads.

Drawn First

The **Drawn First** controls (i.e. outputorder) determine the sequence in which Graphviz draws nodes and edges during rendering. These options are presented as toggle buttons that behave like radio buttons, ensuring that only one drawing order is active at a time. This gives you a quick, intuitive way to adjust whether edges appear above or below nodes—useful when fine-tuning visibility, layering, or stylistic preferences in your diagram.

Output Order Values

Button	Description
breadth	Draws nodes before edges. Edges appear on top and below nodes.
nodes	Nodes are drawn first; edges are drawn afterward. Edges appear on top of nodes.
edges	Edges are drawn first; nodes are drawn afterward, causing nodes to appear on top of edges.

Layout Options

The Algorithm group within the Graphviz tab changes dynamically based upon the layout algorithm chosen. The graph options shown are specific to that particular layout algorithm.

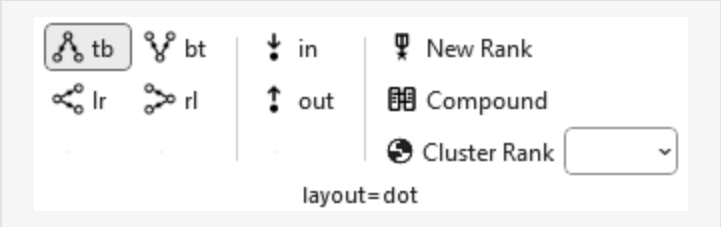
---

layout=circo

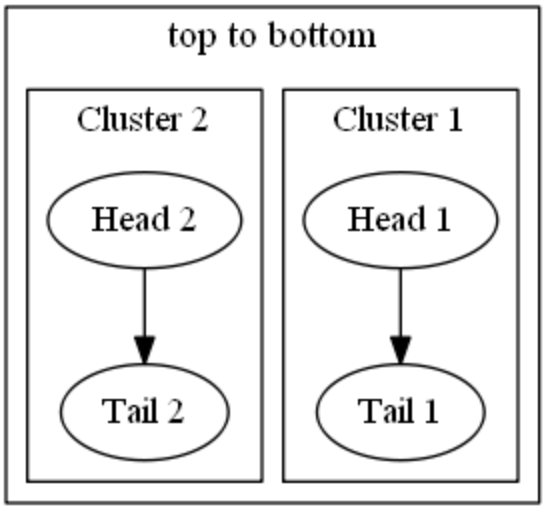
There are no additional dynamic options for `layout=circo`.

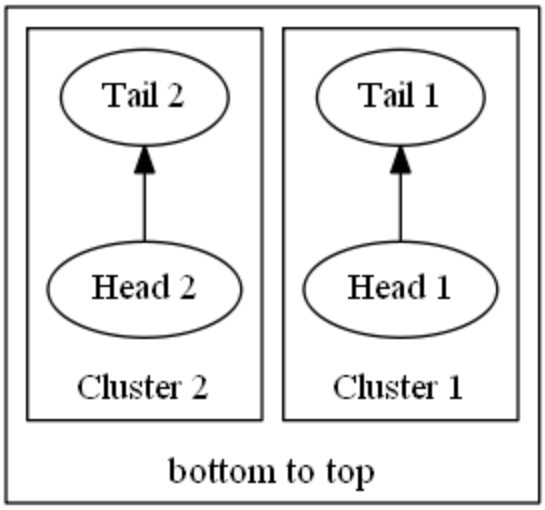
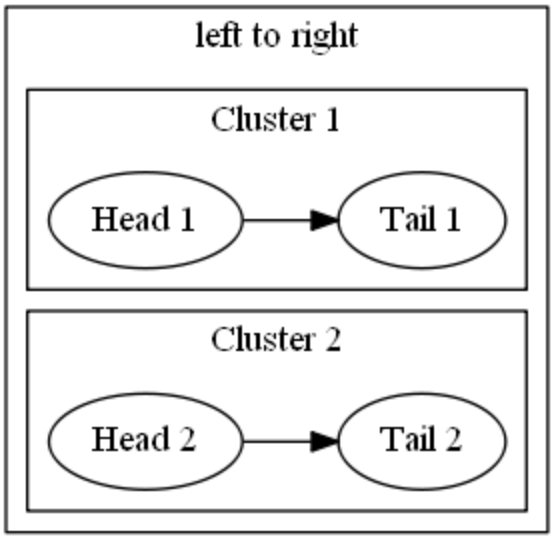
---

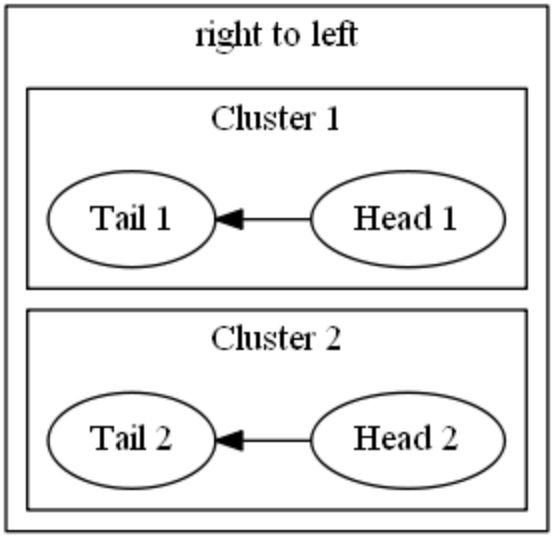
layout=dot



The buttons `[tb]`, `[bt]`, `[lr]`, `[rl]` determine the **Rank Direction** flow of the graph—whether nodes are arranged top-to-bottom, bottom-to-top, left-to-right, or right-to-left. These options are presented as toggle buttons that behave like radio buttons, ensuring that only one direction is active at a time.

Button	Description
tb	Top-to-Bottom. Ranks flow downward (the default for most layouts). 

Button	Description
bt	Bottom-to-Top. Ranks flow upward. 
lr	Left-to-Right. Ranks flow horizontally from left to right. 
rl	Right-to-Left. Ranks flow horizontally from right to left.

Button	Description
	

The `[in]`, `[out]` Ordering buttons determine how edges are arranged around each node during layout.

Button	Description
in	Preserves the order of incoming edges around each node.
out	Preserves the order of outgoing edges around each node.

The **New Rank** button determines how Graphviz handles ranking when clusters are present in the graph. This option is presented as a toggle that behaves like a radio-style switch, ensuring the feature is either fully enabled or disabled. Turning it on allows Graphviz to compute a single global ranking across all clusters, while turning it off preserves the traditional recursive ranking inside each cluster. This gives you a quick, intuitive way to influence how tightly or loosely clustered subgraphs interact in the final layout.

The **Compound** button determines whether edges are allowed to connect into and out of clusters when using layout engines that support this feature. This option is presented as a simple toggle that behaves like a radio-style switch, enabling or disabling compound edge routing. Turning it on allows edges to attach to cluster boundaries using lhead and ltail,

giving you a quick, intuitive way to create more expressive diagrams where relationships span across grouped subgraphs.

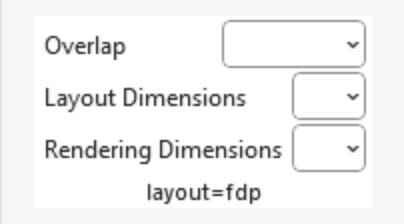
Value	Description
false	Disables compound edges. Edges cannot connect into or out of clusters using `lhead` or `ltail`.
true	Enables compound edges, allowing edges to enter or leave clusters and attach to cluster boundaries.

The **Cluster Rank** control determines how Graphviz ranks clusters relative to one another during layout.

Value	Description
local	Each cluster is ranked independently. This preserves the traditional recursive ranking behavior and often produces compact cluster layouts.
global	All clusters participate in a single, unified ranking. This can create more consistent alignment across clusters but may increase spacing.

---

layout=fdp



The **Overlap** control is presented as a dropdown list that lets you choose how Graphviz handles node collisions during layout. Each option corresponds to a specific overlap-removal strategy supported by Graphviz, ranging from allowing overlaps for speed to applying more advanced algorithms for cleaner spacing.

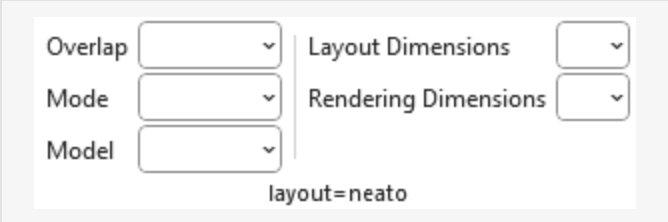
Value	Description
compress	Reduces whitespace by compressing the layout after overlap removal, producing a tighter diagram.
prism	Uses a stress-based algorithm to separate overlapping nodes while preserving layout structure.
scale	Uniformly scales the entire layout until nodes no longer overlap.
scalexy	Scales the layout independently in the X and Y directions to eliminate overlaps.
Voronoi	Uses a Voronoi-based algorithm to push nodes apart by expanding their regions until overlaps are resolved.

The **Layout Dimensions** control (`dim=` attribute) sets the number of dimensions Graphviz uses when computing node positions for certain layout engines (primarily neato, fdp, and sfdp). This option is presented as a dropdown list, allowing you to choose how many dimensions the layout solver operates in. Higher dimensions can help the solver escape local minima and produce cleaner layouts, even though the final output is always projected back into 2D.

The **Rendering Dimensions** control (`dimen=` attribute) specifies how many dimensions are used when interpreting node size attributes such as width, height, and size. This option is presented as a dropdown list, allowing you to choose whether nodes are sized in two dimensions or in higher-dimensional space. Although the final drawing is always 2D, increasing the dimensionality can influence how Graphviz interprets size constraints during layout, giving you a simple, intuitive way to adjust how strictly node size attributes are applied.

---

layout=neato



The **Overlap** control is presented as a dropdown list that lets you choose how Graphviz handles node collisions during layout. Each option corresponds to a specific overlap-removal strategy supported by Graphviz, ranging from allowing overlaps for speed to applying more advanced algorithms for cleaner spacing.

Value	Description
compress	Reduces whitespace by compressing the layout after overlap removal, producing a tighter diagram.
prism	Uses a stress-based algorithm to separate overlapping nodes while preserving layout structure.
scale	Uniformly scales the entire layout until nodes no longer overlap.
scalexy	Scales the layout independently in the X and Y directions to eliminate overlaps.
Voronoi	Uses a Voronoi-based algorithm to push nodes apart by expanding their regions until overlaps are resolved.

The **Mode** control selects the algorithm that Neato uses to compute node positions during layout. This option is presented as a dropdown list, allowing you to choose among several solver strategies that influence how distances, forces, and constraints are optimized. Each mode offers a different balance of speed, stability, and layout style, giving you a quick, intuitive way to experiment with how the underlying algorithm shapes the structure of your diagram.

Value	Description
major	Uses stress majorization to iteratively refine node positions; stable and widely used.
KK	Uses the Kamada–Kawai spring model, optimizing ideal edge lengths through gradient descent.
hier	Produces a top-down, hierarchy-influenced layout similar to dot but using Neato's solver.
ipsep	Applies iterative penalty separation to enforce minimum distances between nodes.

Value	Description
spring	Uses a classical spring-embedder approach for force-directed placement.
maxent	Uses a maximum-entropy–inspired solver to spread nodes evenly while respecting constraints.

The **Model** control selects how Neato interprets edge relationships when computing ideal node distances. This option is presented as a dropdown list, allowing you to choose among several distance-calculation models that influence clustering, separation, and overall layout behavior. Each model offers a different way of translating graph structure into geometric constraints, giving you a quick, intuitive way to shape how Neato arranges your diagram.

The **Layout Dimensions** control (`dim=` attribute) sets the number of dimensions Graphviz uses when computing node positions for certain layout engines (primarily neato, fdp, and sfdp). This option is presented as a dropdown list, allowing you to choose how many dimensions the layout solver operates in. Higher dimensions can help the solver escape local minima and produce cleaner layouts, even though the final output is always projected back into 2D.

The **Rendering Dimensions** control (`dimen=` attribute) specifies how many dimensions are used when interpreting node size attributes such as width, height, and size. This option is presented as a dropdown list, allowing you to choose whether nodes are sized in two dimensions or in higher-dimensional space. Although the final drawing is always 2D, increasing the dimensionality can influence how Graphviz interprets size constraints during layout, giving you a simple, intuitive way to adjust how strictly node size attributes are applied.

---

layout=osage

There are no additional dynamic options for `layout=osage`.

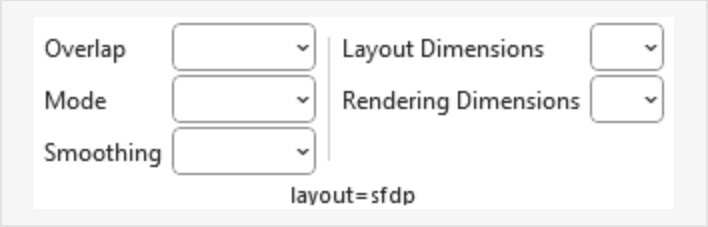
---

layout=patchwork

There are no additional dynamic options for `layout=patchwork`.

---

layout=sfdp



The **Overlap** control is presented as a dropdown list that lets you choose how Graphviz handles node collisions during layout. Each option corresponds to a specific overlap-removal strategy supported by Graphviz, ranging from allowing overlaps for speed to applying more advanced algorithms for cleaner spacing.

Value	Description
compress	Reduces whitespace by compressing the layout after overlap removal, producing a tighter diagram.
prism	Uses a stress-based algorithm to separate overlapping nodes while preserving layout structure.
scale	Uniformly scales the entire layout until nodes no longer overlap.
scalexy	Scales the layout independently in the X and Y directions to eliminate overlaps.
Voronoi	Uses a Voronoi-based algorithm to push nodes apart by expanding their regions until overlaps are resolved.

The **Mode** control selects the algorithm that Neato uses to compute node positions during layout. This option is presented as a dropdown list, allowing you to choose among several solver strategies that influence how distances, forces, and constraints are optimized. Each mode offers a different balance of speed, stability, and layout style, giving you a quick, intuitive way to experiment with how the underlying algorithm shapes the structure of your diagram.

Value	Description
major	Uses stress majorization to iteratively refine node positions; stable and widely used.
KK	Uses the Kamada–Kawai spring model, optimizing ideal edge lengths through gradient descent.
hier	Produces a top-down, hierarchy-influenced layout similar to dot but using Neato's solver.
ipsep	Applies iterative penalty separation to enforce minimum distances between nodes.
spring	Uses a classical spring-embedder approach for force-directed placement.
maxent	Uses a maximum-entropy–inspired solver to spread nodes evenly while respecting constraints.

The Smoothing control is presented as a dropdown list that lets you choose how Graphviz refines the raw node positions produced by the layout engine. Each option applies a different post-processing technique to "smooth out" irregularities, reduce jitter, or improve geometric consistency. This gives you a simple, intuitive way to fine-tune the visual polish of your diagram without altering the underlying layout structure.

Value	Description
none	No smoothing applied. Uses the raw layout positions exactly as computed.
avg_dist	Adjusts node positions based on average distances to neighbors, reducing local irregularities.
graph_dist	Smooths positions using graph-theoretic distances, improving global consistency.
power_dist	Applies a power-law weighting to distances, emphasizing stronger relationships.
rng	Uses a Relative Neighborhood Graph–based smoothing to reduce noise while preserving structure.
spring	Applies a light spring-embedder pass to gently relax node positions.
triangle	Uses triangle-based geometric smoothing to even out spacing in dense regions.

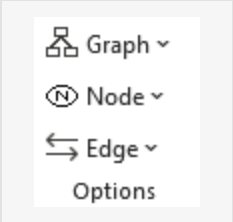
The **Layout Dimensions** control (`dim=` attribute) sets the number of dimensions Graphviz uses when computing node positions for certain layout engines (primarily neato, fdp, and sfdp). This option is presented as a dropdown list, allowing you to choose how many dimensions the layout solver operates in. Higher dimensions can help the solver escape local minima and produce cleaner layouts, even though the final output is always projected back into 2D.

The **Rendering Dimensions** control (`dimen=` attribute) specifies how many dimensions are used when interpreting node size attributes such as width, height, and size. This option is presented as a dropdown list, allowing you to choose whether nodes are sized in two dimensions or in higher-dimensional space. Although the final drawing is always 2D, increasing the dimensionality can influence how Graphviz interprets size constraints during layout, giving you a simple, intuitive way to adjust how strictly node size attributes are applied.

Layout = `twopi`

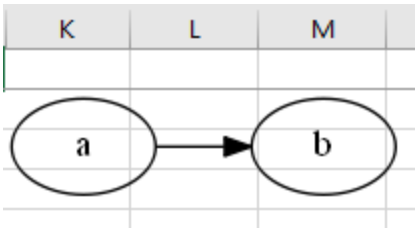
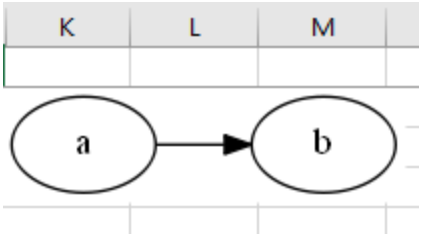
There are no additional dynamic options for `layout=patchwork`.

Options



Graph

Optional attributes which can be checked for inclusion in the Graphviz source. These attributes have graph-level scope.

Label	Control Type	Description
Drawing		
Center Drawing	Checkbox	Checking this item will cause the graph to be centered in the page, assuming the graph is smaller than the page size.
Force xlabel placement	Checkbox	If checked, all `xlabel` attributes are placed, even if there is some overlap with nodes or other labels.
Rotate 90 counterclockwise	Checkbox	If checked, causes the final layout to be rotated counterclockwise by 90 degrees.
Transparent Background	Checkbox	Toggles the background color between white and transparent. Transparent backgrounds are useful if you intend to layer the graphs in an image editor or paste them into a Microsoft Word document. **Choices:** • *Checked* - Background is transparent  • *Unchecked* - Graph background is white. 

Label	Control Type	Description
		Note: It is possible to set the graph background color to any valid color by specifying the `bgcolor=` attribute as a graph option on the `settings` worksheet.
Include image path	Checkbox	If checked, adds the `imagepath` attribute to the graph. **Choices:** • *Checked* - Path to the images is added. • *Unchecked* - Path to images is omitted.

Node

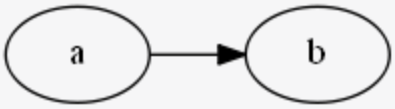
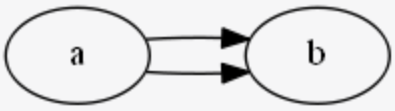
Choices which control which nodes are included in the Graphviz source, and how the labels should be represented.

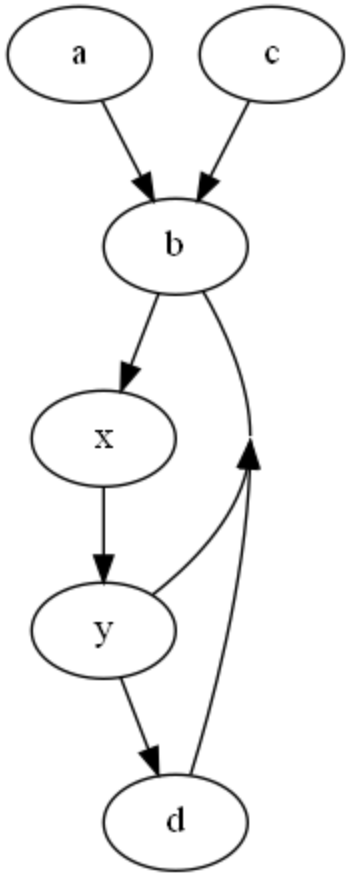
Label	Control Type	Description
Filter		
Include stand-alone nodes	Checkbox	Include or exclude nodes without relationships (i.e., island nodes). When using views to exclude relationship edges there may be nodes left in the diagram that are not connected to anything. This setting specifies if these island nodes should be included or excluded from the diagram. **Choices:** • *Checked* - retain the island nodes • *Unchecked* - drop the island nodes from the diagram
Label Columns		

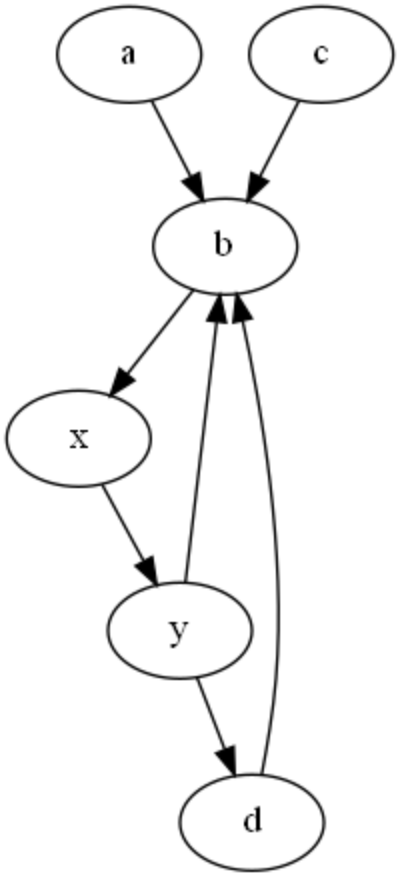
Label	Control Type	Description
Include `Label`	Checkbox	Include or exclude Labels column data? Allows you to turn labels on/off in the graph. **Choices:** • *Checked* - Include Label column data • *Unchecked* - Drop the Label column data from the graph
Include `External Label`	Checkbox	Include or exclude External Labels column data? Allows you to turn outside (xlabel) labels on/off in the graph. **Choices:** • *Checked* - Include External Label column data • *Unchecked* - Drop the External Label column data from the graph
Label Values		
When the `Label` column is blank...	Menu	Include or exclude blank values in the Label column? When the `Label` column is blank on the data worksheet on a row which refers to a node it can mean two possible things. One interpretation is to remove the label from the node, as might be useful when using images to represent nodes. The other interpretation is to let the graph default to displaying the value in the `Item` column. **Choices:** • `...use blank for the node label` - use a blank label as the node's label text • `...use the node identifier as the label` - show the value in the `Item` column as the label text

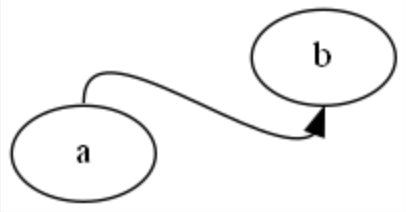
Edge

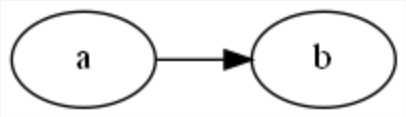
Choices which control how edges should be specified in the Graphviz source, and how the edge labels should be represented.

Label	Control Type	Description
Consolidate		
Apply "strict" rules	Checkbox	Specifies the strict attribute for the top-level graph. Describing the graph as strict forbids the creation of multi-edges, i.e., there can be at most one edge with a given tail node and head node in the directed case. For undirected graphs, there can be at most one edge connected to the same two nodes. Subsequent edge statements using the same two nodes will identify the edge with the previously defined one and apply any attributes given in the edge statement. **Choices:** • *Checked* - Includes the strict attribute  Edges have been consolidated. • *Unchecked* - Omits the strict attribute  Edges have not been consolidated.
Concentrate edges	Checkbox	If checked, use edge concentrators. This merges multi-edges into a single edge and causes partially parallel edges to share part of their paths. This feature is only available if the layout algorithm is **dot**.

Label	Control Type	Description
		Choices: • *Checked* - Include the concentrate attribute  Edges have been concentrated • *Unchecked* - Omits the concentrate attribute

Label	Control Type	Description
		 Edges are not concentrated
Filter		
Include edges which reference undefined nodes	Checkbox	Include/Exclude relationships Include stand-alone edges (i.e., orphan edges). When using views to exclude nodes there may be un-styled nodes included in the diagram due to edge references. This setting specifies if the edges should be included or excluded from the diagram. **Choices:** • *Checked* - retain edges which have references to undefined nodes

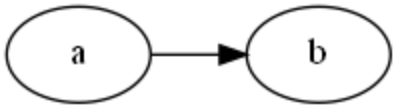
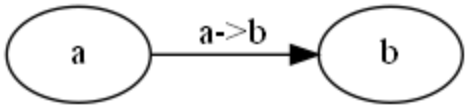
<table>
<tr><th>Label</th><th>Control Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>Unchecked - drop any edges which do not refer to defined nodes</td></tr>
<tr><td>Include Ports</td><td>Checkbox</td><td>Retain/Remove port values from the nodes in an edge relationship. Given:<table><tr><th>Item</th><th>Related Item</th></tr><tr><td>a:n</td><td>b:s</td></tr></table>Choices:
Checked - retain the ports when creating the edge syntax. 

<code>a:n -> b:s</code>
Unchecked - removes the ports specified when creating the edge syntax. 

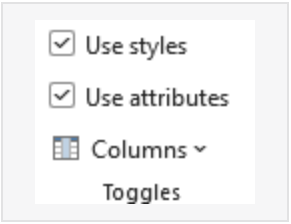
<code>a -> b</code></td></tr>
<tr><td>Label Columns</td><td></td><td></td></tr>
<tr><td>Include <code>Label</code></td><td>Checkbox</td><td>Include or exclude Labels column data? Allows you to turn edge labels on/off in the graph.

Choices:
Checked - Include Label column data
Unchecked - Omit the Label column data from the graph</td></tr>
</table>

Label	Control Type	Description
Include `External Label`		Include or exclude External Labels column data? Allows you to turn outside (xlabel) edge labels on/off in the graph. **Choices:** • *Checked* - Include External Label column data • *Unchecked* - Omit the External Label column data from the graph
Include `Head Label`	Checkbox	Include or exclude Head Labels column data? Allows you to turn edge head labels on/off in the graph. **Choices:** • *Checked* - Include Head Label column data • *Unchecked* - Omit the Head Label column data from the graph
Include `Tail Label`	Checkbox	Include or exclude Tail Labels column data? Allows you to turn edge tail labels on/off in the graph. **Choices:** • *Checked* - Include Tail Label column data • *Unchecked* - Omit the Table Label column data from the graph
Label Values		
When the `Label` column is blank...	Menu	Include or exclude blank values in the Label column? When the `Label` column is blank on the data worksheet on a row which refers to an edge it can mean two possible things. One interpretation is to remove the label from the edge. The other interpretation is to let the graph default to displaying the

Label	Control Type	Description
		value Graphviz assigns to the edge relationship. **Choices:** • `...the label is blank` - use the blank label as the node's label text  • `...use the edge name as the label` - show the value in the `Item` column as the label text 

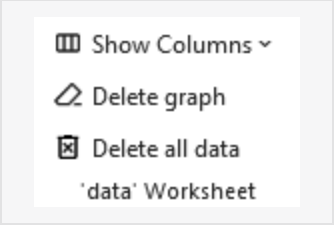
Toggles



Label	Control Type	Description
Use styles	Checkbox	Specifies if the style attributes associated with the Style Name assigned to a node, edge, or cluster should be used when the graph is generated. **Choices:** • *Checked* - use the style format

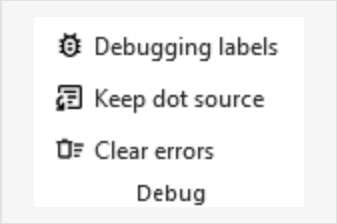
Label	Control Type	Description
		• *Unchecked* - do not use the style format (i.e., use default Graphviz rendering method)
Use attributes	Checkbox	Specifies if the `Attributes` style attributes on the `data` worksheet should be included or omitted when the graph is generated. **Choices:** • *Checked* - include the style attributes • *Unchecked* - do not include the style attributes
Columns	Dropdown List	A list of column names on the `data` worksheet which can be displayed or hidden. **Choices:** • *Checked* - show the column • *Unchecked* - hide the column

'data' Worksheet

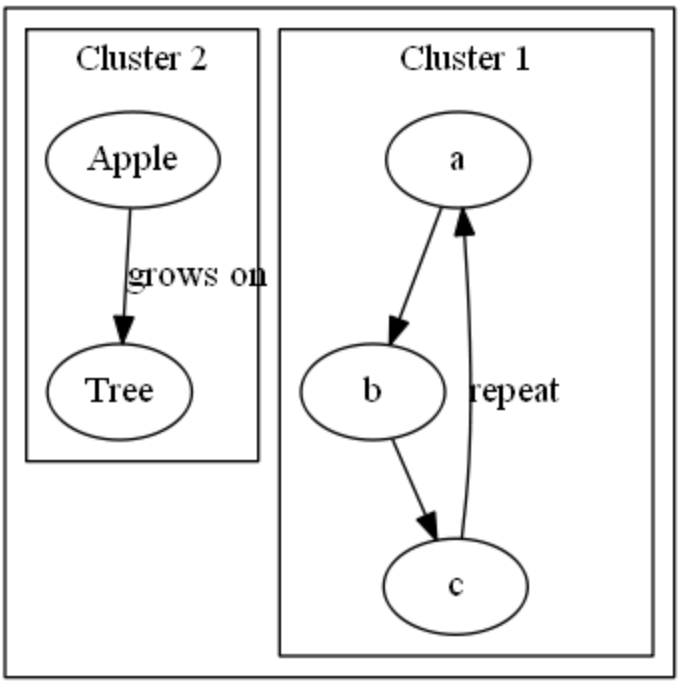
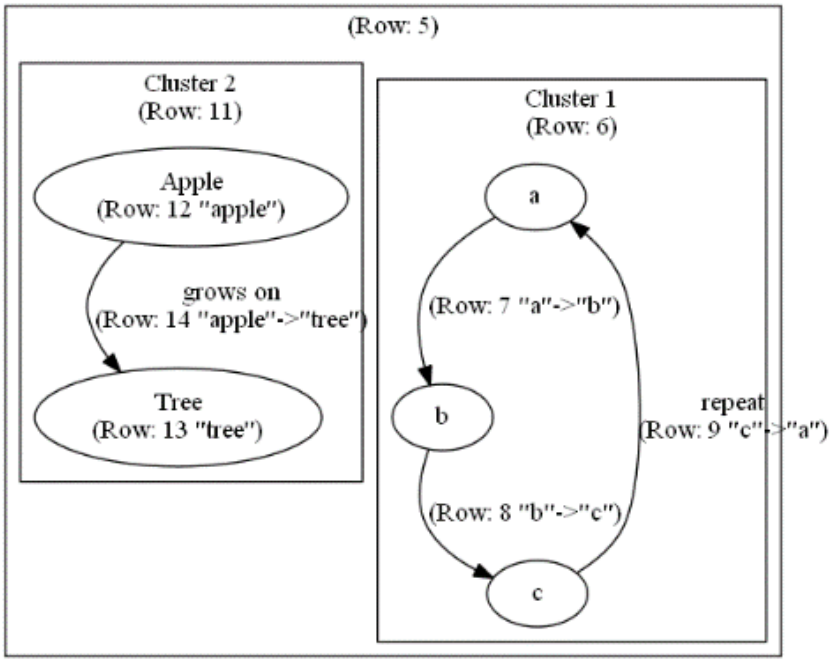


Label	Control Type	Description
Show Columns	Menu	Displays a list of all the columns used by the data worksheet. Allows you to show or hide columns by clicking of the column names. Checked columns are shown, unchecked columns are hidden
Delete Graph	Button	Clicking on this button will delete the graph from the worksheet. This is useful when adding rows as new rows will stretch the image. You may also find you want to delete the image before saving the file to reduce the file size.
Delete all data	Button	Resets the `data` worksheet to blank cells, and deletes any graphs if present.

Debug

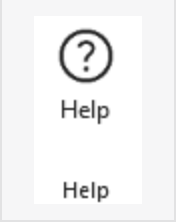


Label	Control Type	Description
Debugging labels	Checkbox	Turning this option to `on` causes additional information such as the row number and Item identifiers to be included in the labels of nodes, edges, and clusters. **Choices:** • *Unchecked* - Do not add information to the labels • *Checked* - Add information to the labels Unchecked

Label	Control Type	Description
		 Checked 
Keep dot source	Checkbox	Specifies what should be done with the text file sent to Graphviz after the graphing step is complete when `Graph to File` is used

Label	Control Type	Description
		to create the graph. **Choices:** • *Checked* - retain the file. It will be in the same directory as the graph file with the same file name except for the file extension (which will be `.gv`). • *Unchecked* - delete the file
Clear errors	Button	Resets the error message column

Help



Provides the `Help` content for the `Graphviz` ribbon tab.

Label	Control Type	Description
Help	Button	Provides a link to this web page.

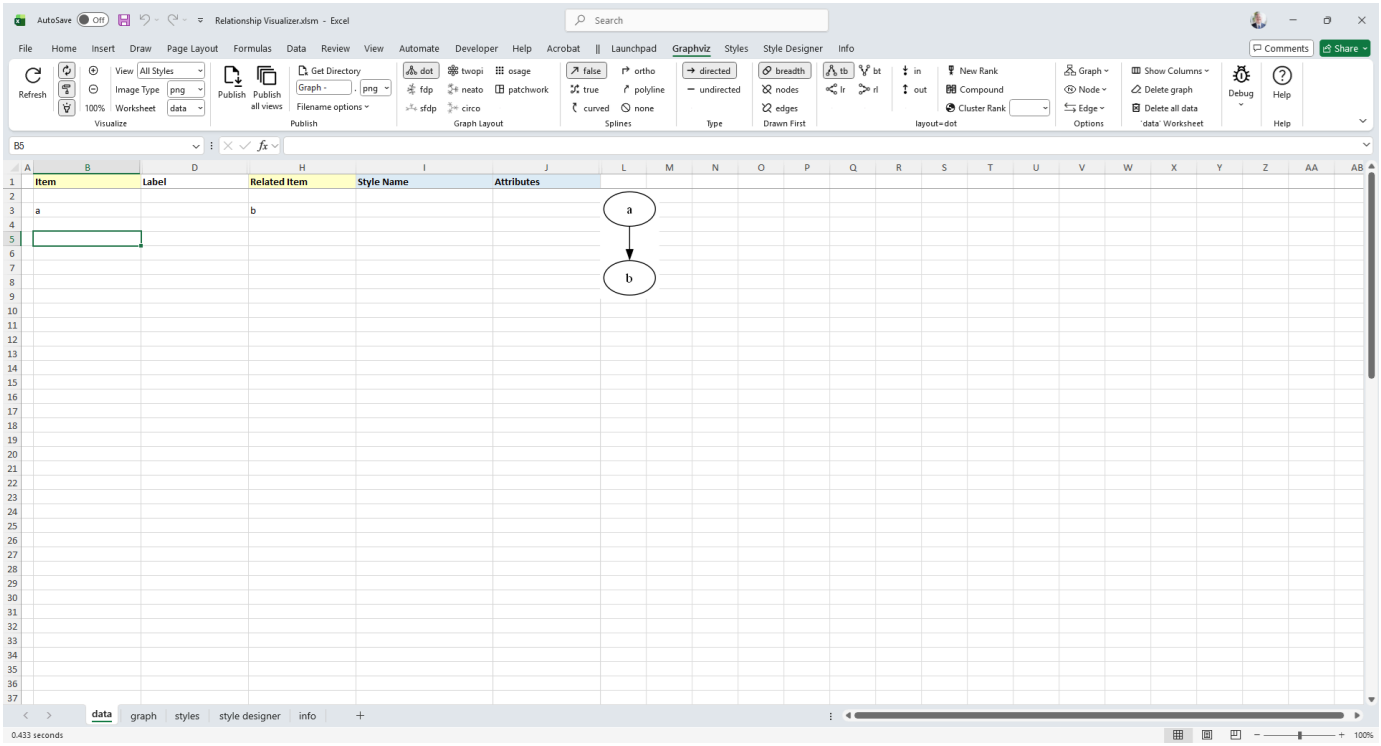
Core Concepts

The simplest way to draw a graph is to place values in the `Item` and the `Related Item` columns.

For our first graph, we will draw an *'a' is related to 'b'* relationship.

1. Click on the `Graphviz` ribbon tab to activate it (if it is not the current active tab)
2. Enable the `Automatic` toggle button (if it is not already checked)
3. Ensure the `Workbook` dropdown is set to `data`, which will cause the graph to be displayed within the data worksheet.
4. In row 3 type 'a' in the `Item` column, and 'b' in the `Related Item` column. The result will be drawn beside the data as you change cells.

The results should resemble the following example:



Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b;
}
```

Congratulations, you have created your first graph!

TIP

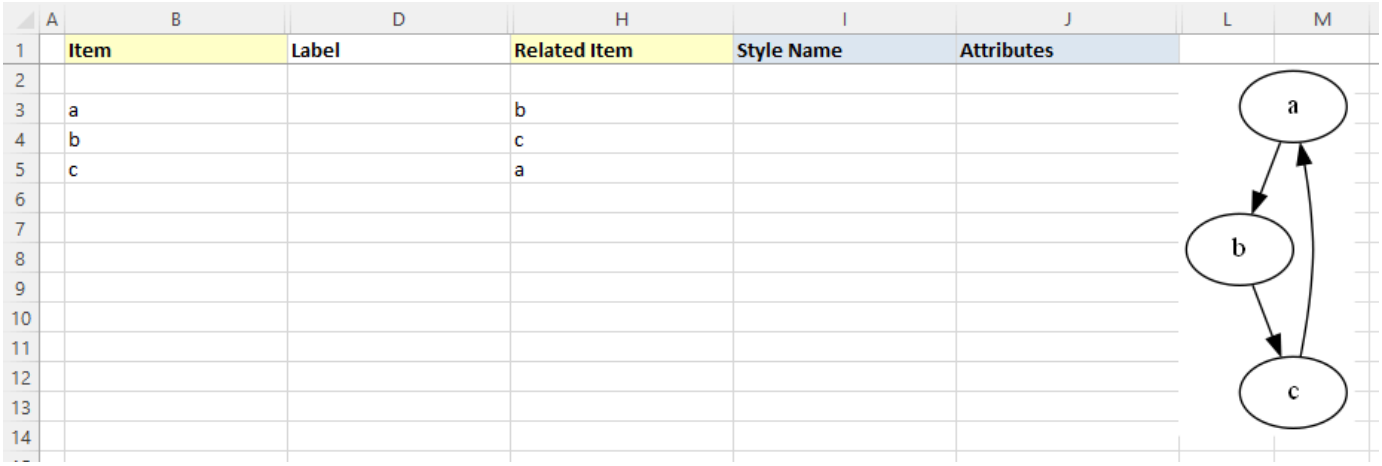
- If the `Automatic` toggle button is enabled the graph will draw as data is entered into each cell.
- If the `Automatic` toggle button is disabled, pressing the `Refresh Graph` button is necessary to draw the graph.

Connect More Items

Next, lets expand upon the graph we just created to have additional relationships. Assume that:

- *'a' is related to 'b'* (already drawn)
- *'b' is related to 'c'*
- *'c' is related to 'a'*

The Excel data appears as shown on rows 3-5. The Excel worksheet now looks like:

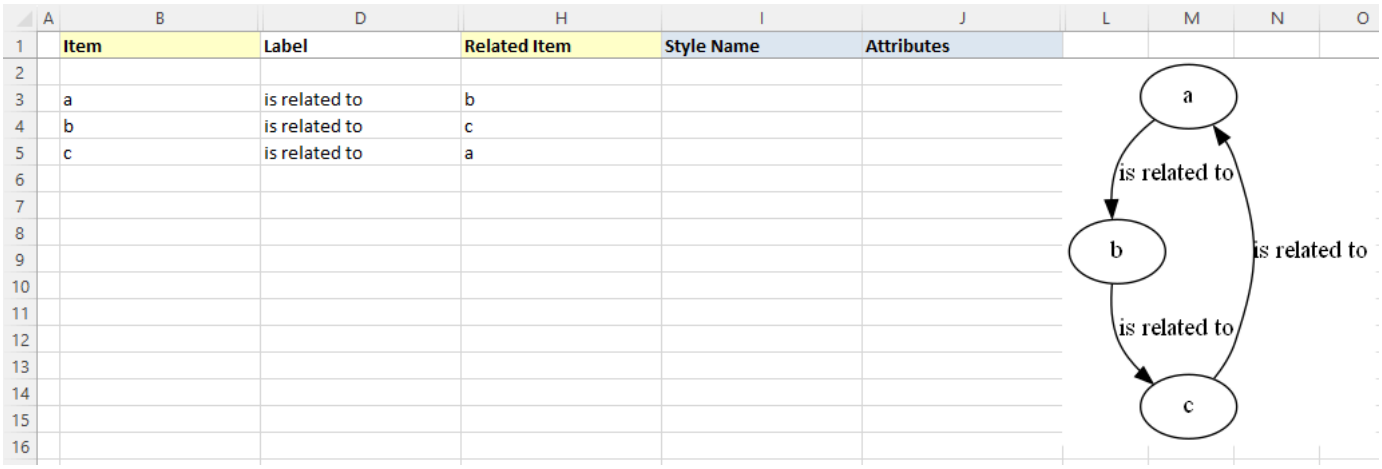


Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b;
    b -> c;
    c -> a;
}
```

Add Edge Labels

Now, let us add data into the `Label` column to label the relationships. Fill in Column D as shown below. Press the `Refresh Graph` button, and the Excel worksheet now looks like:



	A	B	D	H	I	J	L	M	N	O
1		Item	Label	Related Item	Style Name	Attributes				
2										
3		a	is related to	b						
4		b	is related to	c						
5		c	is related to	a						
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										

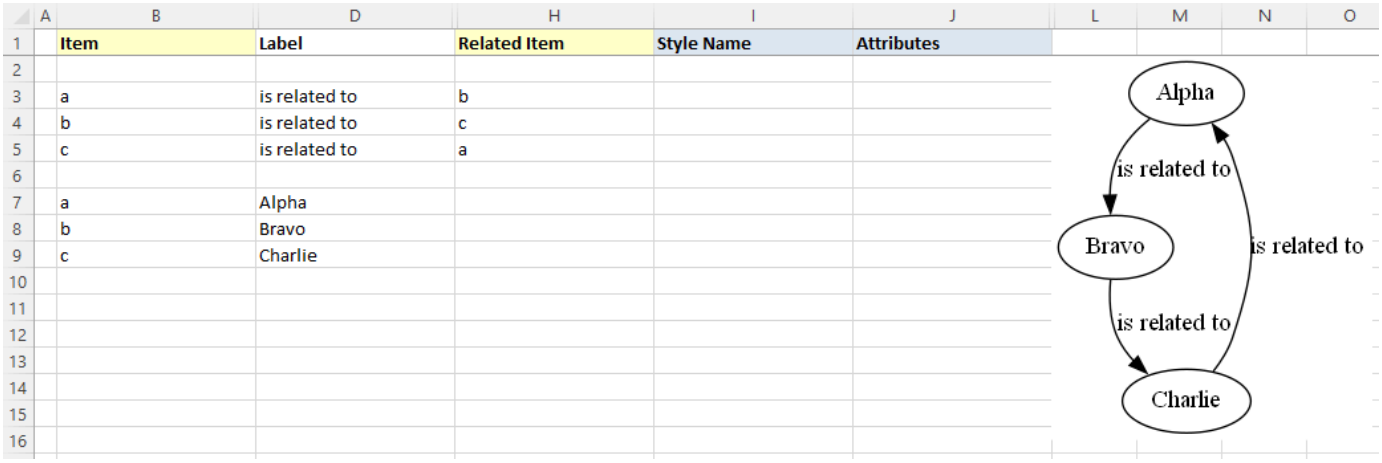
Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b[ label="is related to" ];
    b -> c[ label="is related to" ];
    c -> a[ label="is related to" ];
}
```

Add Node Labels

The graph is how we want to see it, but the nodes need to be labeled. We do not want to change all our edges; however, we would like to replace 'a' with 'Alpha', 'b' with 'Bravo', and 'c' with 'Charlie'. The Relationship Visualizer assumes that when there is information in the `Item` column, but not in the `Related Item` column that the data corresponds to a node.

To label the nodes we will add 3 node definitions to the "data worksheet (rows 6, 7, 8) and press the `Refresh Graph` button. The Excel worksheet now looks like:



	Item	Label	Related Item	Style Name	Attributes
2					
3	a	is related to	b		
4	b	is related to	c		
5	c	is related to	a		
6					
7	a	Alpha			
8	b	Bravo			
9	c	Charlie			
10					
11					
12					
13					
14					
15					
16					

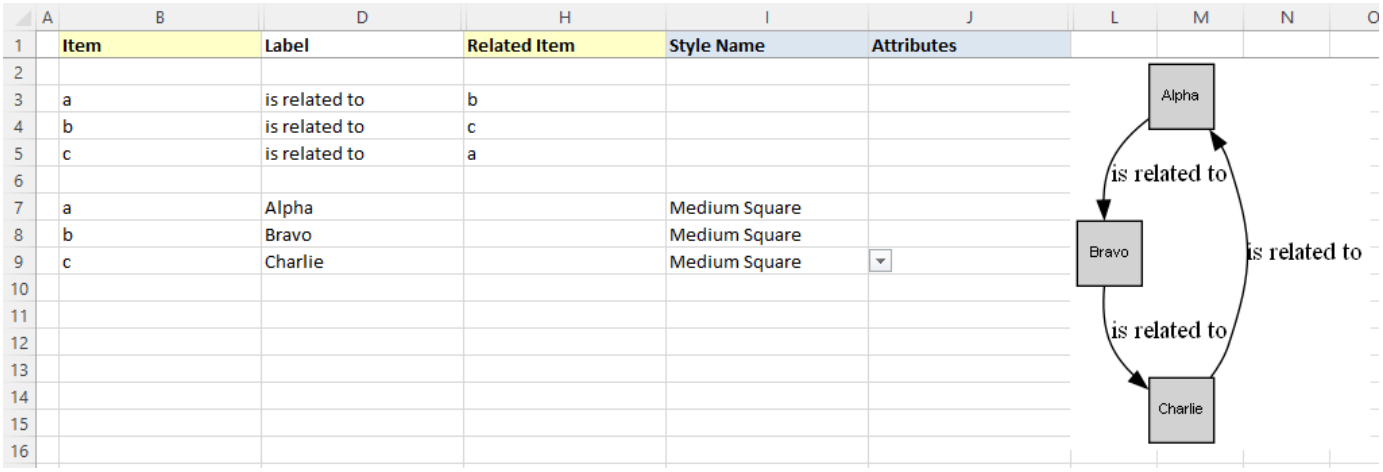
Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b[ label="is related to" ];
    b -> c[ label="is related to" ];
    c -> a[ label="is related to" ];
    a [ label="Alpha" ];
    b [ label="Bravo" ];
    c [ label="Charlie" ];
}
```

Apply a pre-defined node style

Next we will apply a pre-defined style to the nodes. Later on we will learn how to create our own node styles, but for now we will choose one of the default styles provided out of the box.

On rows 7, 8, and 9 tab to the `Style Name` column. A dropdown list will appear. Select the style `Medium Square`. The Excel worksheet now looks like:



	Item	Label	Related Item	Style Name	Attributes
2					
3	a	is related to	b		
4	b	is related to	c		
5	c	is related to	a		
6					
7	a	Alpha		Medium Square	
8	b	Bravo		Medium Square	
9	c	Charlie		Medium Square	
10					
11					
12					
13					
14					
15					
16					

Graphviz Source

```
strict digraph "main"
digraph "Relationship Visualizer"
{
    a -> b[ label="is related to" ];
    b -> c[ label="is related to" ];
    c -> a[ label="is related to" ];
    a [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    b [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    c [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
}
```

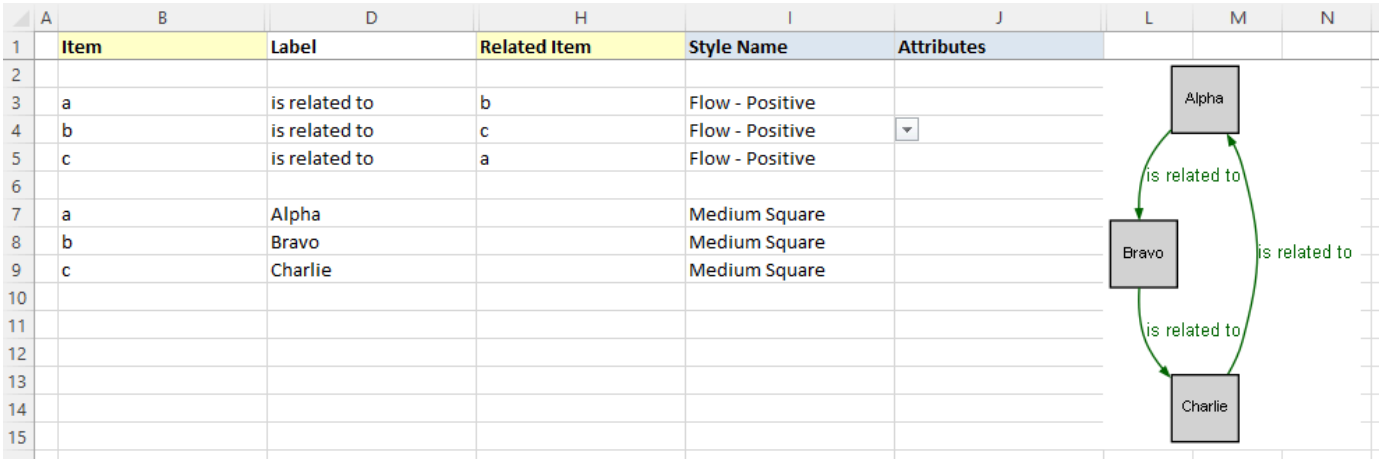
Apply a pre-defined edge style

Next we will apply a pre-defined style to the edges. Later on we will learn how to create our own edge styles, but for now we will choose one of the default styles provided out of the

box.

On rows 3, 4, and 5 move to the `Style Name` column. A dropdown list will appear. Select the style `Flow - Positive`. This style uses the color `dark green`.

The Excel worksheet now looks like:



	Item	Label	Related Item	Style Name	Attributes
3	a	is related to	b	Flow - Positive	
4	b	is related to	c	Flow - Positive	
5	c	is related to	a	Flow - Positive	
7	a	Alpha		Medium Square	
8	b	Bravo		Medium Square	
9	c	Charlie		Medium Square	

Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    b -> c[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    c -> a[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    a [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    b [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    c [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
}
```

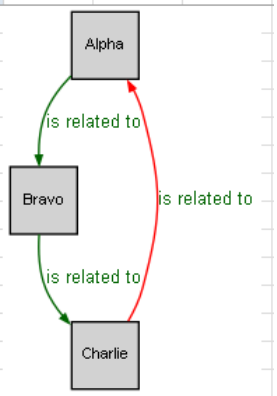
Apply an attribute to an edge

Next we will override the color on one of the edges.

On rows 5 move to the `Attributes` column. Enter the value `color="red"`. The edge color will change from `dark green` to `red`. The font color, however will remain dark green.

The Excel worksheet now looks like:

	A	B	D	H	I	J	L	M	N
1		**Item**	**Label**	**Related Item**	**Style Name**	**Attributes**			
2									
3		a	is related to	b	Flow - Positive				
4		b	is related to	c	Flow - Positive				
5		c	is related to	a	Flow - Positive	color="red"			
6									
7		a	Alpha		Medium Square				
8		b	Bravo		Medium Square				
9		c	Charlie		Medium Square				
10									
11									
12									
13									
14									
15									



Graphviz Source

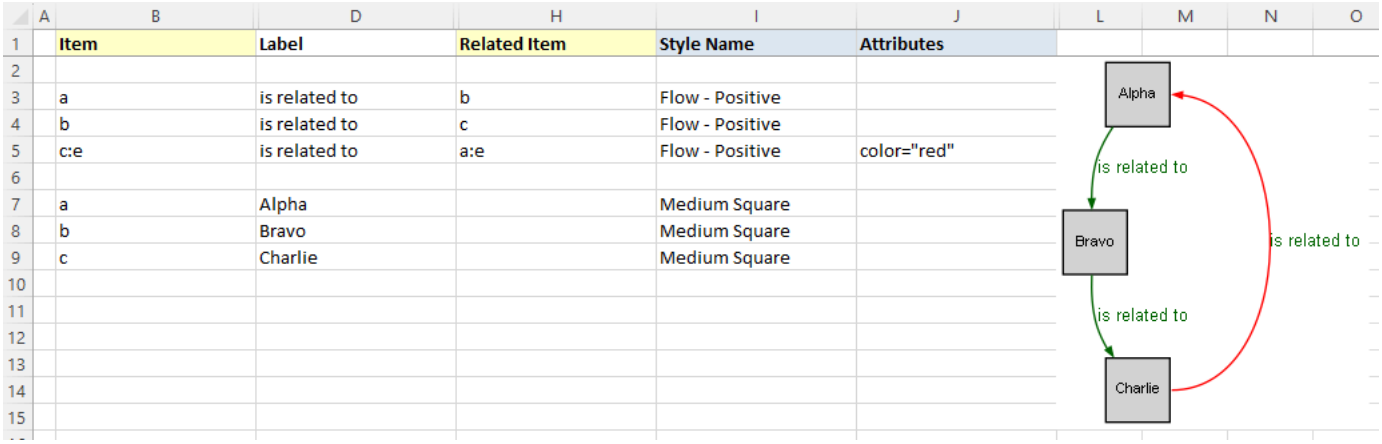
```
digraph "Relationship Visualizer"
{
    a -> b[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    b -> c[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    c -> a[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    a [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    b [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    c [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
}
```

Specify Ports

Graphviz decides what it thinks is the best placement of the head and tail of an edge to produce a balanced graph.

Sometimes you might want to control where the edges begin or end. You can do that by specifying a port on the `Item` or `Related Item` ID, in the same manner as a URL. Ports are identified by a colon character `:` and then a compass point `n`, `s`, `e`, `w`, `ne`, `nw`, `se`, `sw` or `c` for center.

Lets change row 5 from the example above to have the edge from "c" to "a" exit from the east port of "c", and enter the east port of "a". The `Item` is now specified as `c:e`, and the Related Item is specified as `a:e` as shown in row 5. Press the `Refresh Graph` button, and the Excel worksheet now looks like:



	Item	Label	Related Item	Style Name	Attributes
3	a	is related to	b	Flow - Positive	
4	b	is related to	c	Flow - Positive	
5	c:e	is related to	a:e	Flow - Positive	color="red"
7	a	Alpha		Medium Square	
8	b	Bravo		Medium Square	
9	c	Charlie		Medium Square	

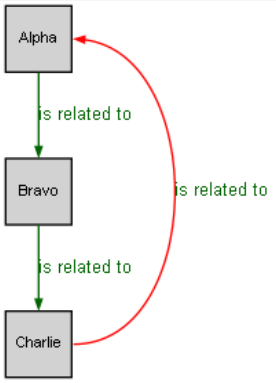
Graphviz Source

```
digraph "Relationship Visualizer"
{
    a -> b[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    b -> c[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    c:e -> a:e[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsi
    a [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    b [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    c [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
}
```

Straighten Edges

Graphviz has a `weight` attribute which tells it to favor straighter lines. Lets add the attribute on rows 3, and 4 to tidy up the diagram. In the `Attributes` column add the value `weight=10`. The graph now appears as:

	Item	Label	Related Item	Style Name	Attributes
3	a	is related to	b	Flow - Positive	weight=10
4	b	is related to	c	Flow - Positive	weight=10
5	c:e	is related to	a:e	Flow - Positive	color="red"
7	a	Alpha		Medium Square	
8	b	Bravo		Medium Square	
9	c	Charlie		Medium Square	



Graphviz Source

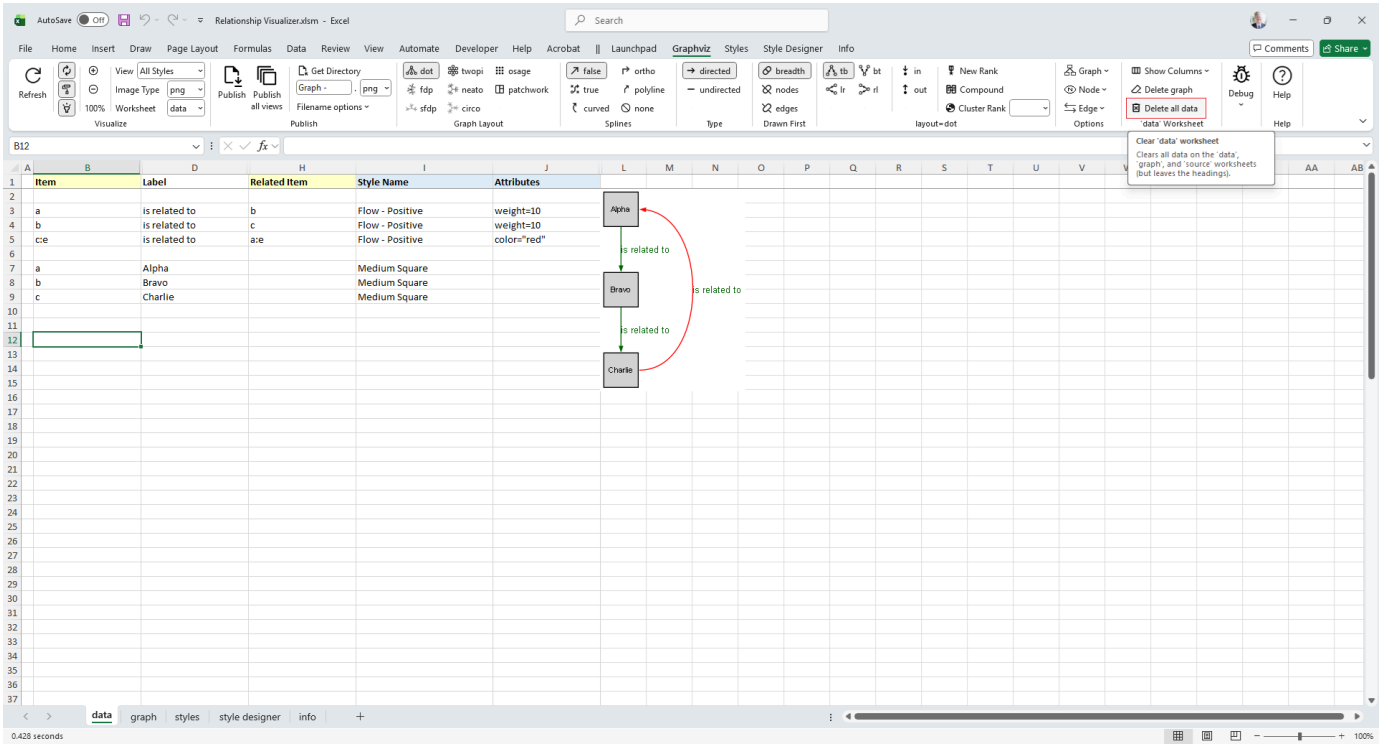
```
digraph "Relationship Visualizer"
{
    a -> b[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    b -> c[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsize=0
    c:e -> a:e[ fontname=Arial fontsize=10 color=darkgreen fontcolor=darkgreen arrowsi
    a [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    b [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
    c [ shape=square height=0.5 width=0.5 fixedsize=True style=filled penwidth=1 fontna
}
```

Delete all data

Lets start by clearing the `data` worksheet so that we can create a new graph with clusters. Click on the `Delete all data` button.

Notice that if you hover the mouse over a Ribbon control a tooltip of help will appear.

Once you click `Delete all data` the `data` worksheet is reset to blank form.



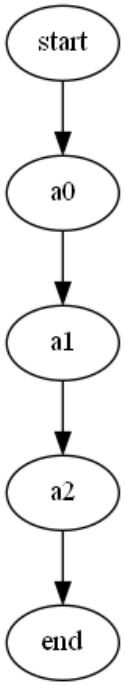
Specify Clusters

With the `data` worksheet cleared, lets create a new graph.

If you wish to cluster some elements of the graph you can do so by adding a row with an open brace "{" in the `Item` column above the first row of data to be placed in the group and provide a title for the cluster in the `Label` column. Next, add row with a close brace "}" in the `Item` column after the last row of data.

For example, this Excel worksheet does not have clusters.

	A	B	D	H	I	J	L
1		Item	Label	Related Item	Style Name	Attributes	
2							
3		start		a0			
4		a0		a1			
5		a1		a2			
6		a2		end			
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							



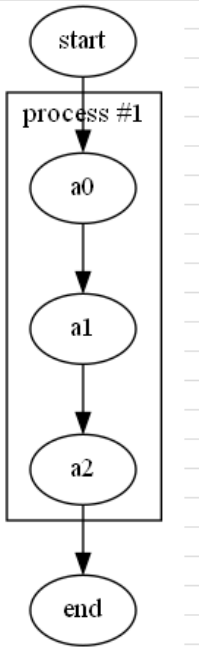
Graphviz Source

```
digraph "Relationship Visualizer"
{
    start -> a0;
    a0 -> a1;
    a1 -> a2;
    a2 -> end;
}
```

To cluster nodes a0, a1, and a2, calling the cluster "process #1" the worksheet is revised to add an open brace {with the label "process #1" on row 3, and a close brace } on rows 6 as follows.

Press the `Refresh Graph` button, and the Excel worksheet now looks like:

	Item	Label	Related Item	Style Name	Attributes
3	start		a0		
4	{	process #1			
5	a0		a1		
6	a1		a2		
7	}				
8	a2		end		



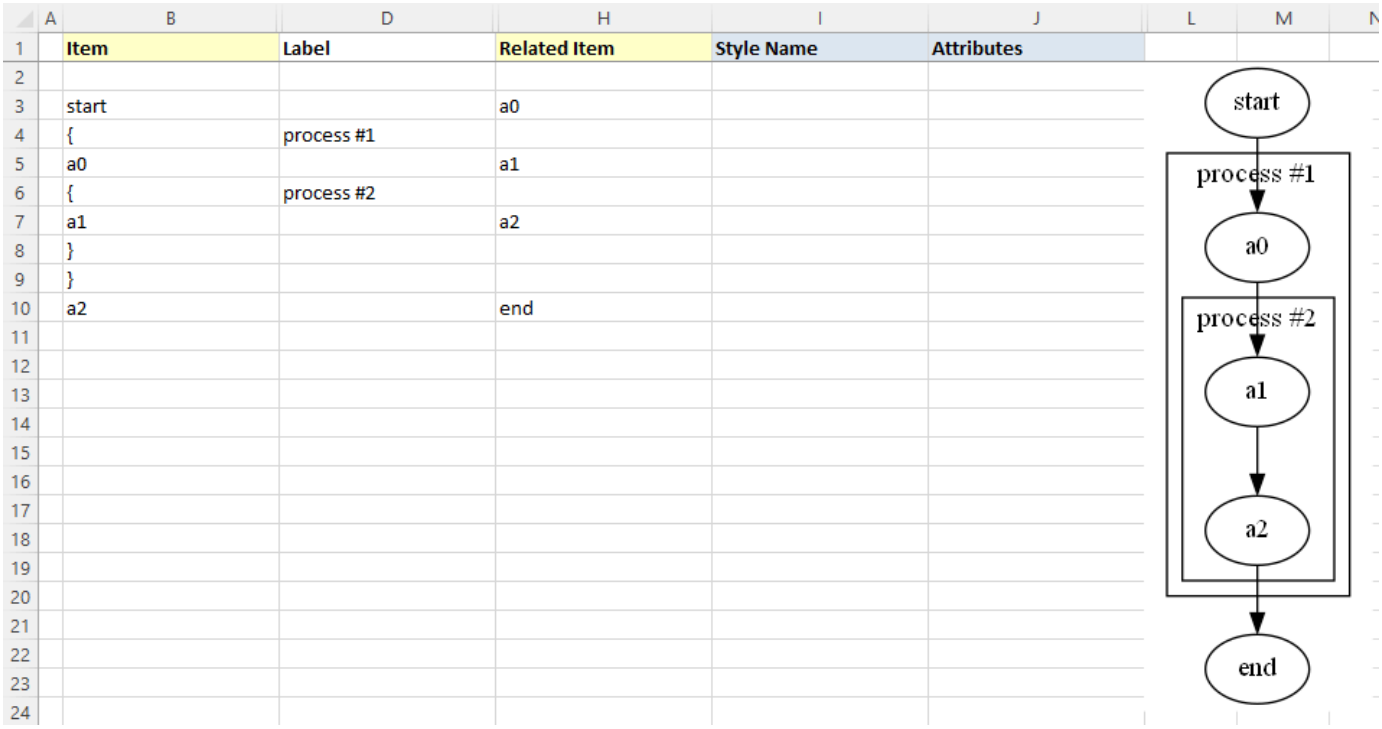
Graphviz Source

```
digraph "Relationship Visualizer"
{
    start -> a0;
    subgraph "cluster_1" {  label="process #1"
        a0 -> a1;
        a1 -> a2;
    }
    a2 -> end;
}
```

Specify Clusters Within Clusters

Graphviz permits clusters within clusters. Let us extend the example by adding an additional set of braces to cluster the relationship between a1 and a2. We will insert a new row 5 placing an open brace { in the `Item` column with the Label column set to "process #2", and a new row 7 with a close brace } in the `Item` column.

Press the `Refresh Graph` button, and the Excel worksheet now looks like:



	Item	Label	Related Item	Style Name	Attributes
2					
3	start		a0		
4	{	process #1			
5	a0		a1		
6	{	process #2			
7	a1		a2		
8	}				
9	}				
10	a2		end		

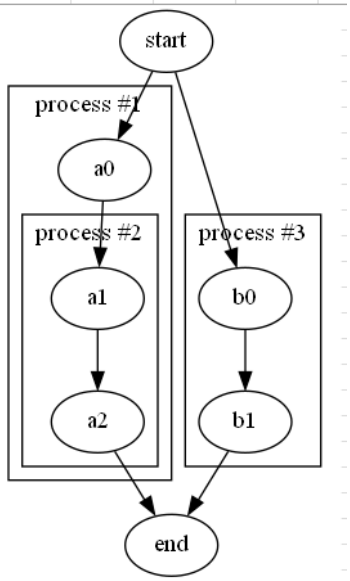
Graphviz Source

```
digraph "Relationship Visualizer"
{
    start -> a0;
    subgraph "cluster_1" {  label="process #1"
        a0 -> a1;
        subgraph "cluster_2" {  label="process #2"
            a1 -> a2;
        }
    }
    a2 -> end;
}
```

Graphviz does not limit the number of clusters you can have. In this example, we have added rows 10-14 to insert an additional cluster labeled "process #3".

Press the `Refresh Graph` button, and the Excel worksheet now looks like:

	Item	Label	Related Item	Style Name	Attributes
2					
3	start		a0		
4	{	process #1			
5	a0		a1		
6	{	process #2			
7	a1		a2		
8	}				
9	}				
10	a2		end		
11					
12	start		b0		
13	{	process #3			
14	b0		b1		
15	}				
16	b1		end		



Graphviz Source

```
digraph "Relationship Visualizer"
{
    start -> a0;
    subgraph "cluster_1" {  label="process #1"
        a0 -> a1;
        subgraph "cluster_2" {  label="process #2"
            a1 -> a2;
        }
    }
    a2 -> end;
    start -> b0;
    subgraph "cluster_3" {  label="process #3"
        b0 -> b1;
    }
    b1 -> end;
}
```

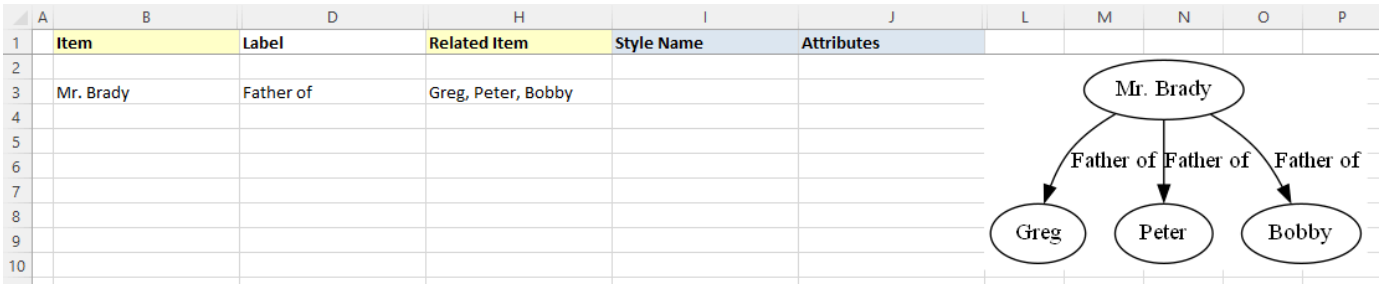
What is important to note is that you must ensure that you have an equal number of open braces as you do close braces.

WARNING

Graphviz will not draw the graph if there is a mismatch between the number of open `{` and close `}` braces.

Specify Comma-separated Items

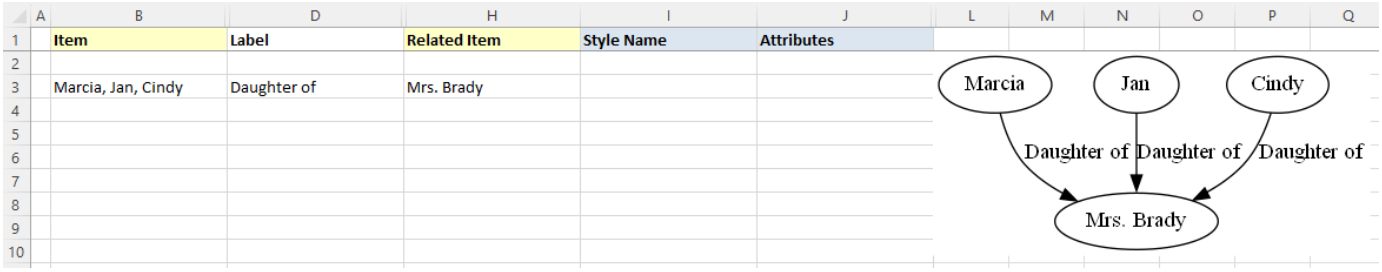
Another feature of the Relationship Visualizer is the ability to specify a comma-separated list of Item names and have a relationship created for each Item. For example, we can say that Mr. Brady is the father of Greg, Peter, and Bobby on one row as follows:



Graphviz Source

```
digraph "Relationship Visualizer"
{
    "Mr. Brady" -> Greg[ label="Father of" ];
    "Mr. Brady" -> Peter[ label="Father of" ];
    "Mr. Brady" -> Bobby[ label="Father of" ];
}
```

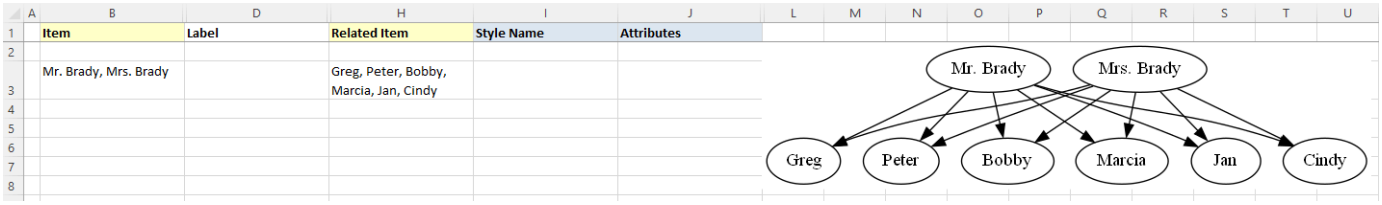
The comma-separated list can also appear in the `Item` column, such as:



Graphviz Source

```
digraph "Relationship Visualizer"
{
    Marcia -> "Mrs. Brady"[ label="Daughter of" ];
    Jan -> "Mrs. Brady"[ label="Daughter of" ];
    Cindy -> "Mrs. Brady"[ label="Daughter of" ];
}
```

Or a comma-separated list can be used in both the `Item`, and the `Related Item` column such as the parental relationship below:



Item	Label	Related Item	Style Name	Attributes
Mr. Brady, Mrs. Brady		Greg, Peter, Bobby, Marcia, Jan, Cindy		

Graphviz Source

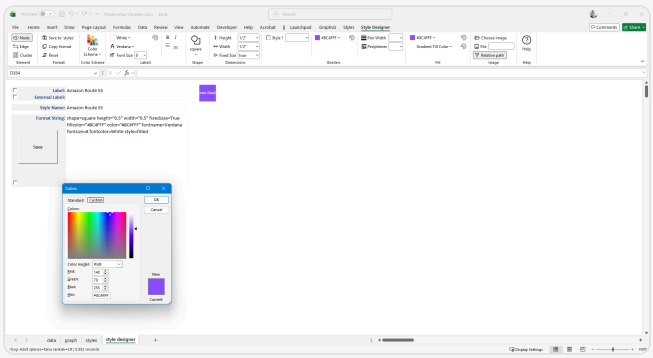
```
digraph "Relationship Visualizer"
{
    "Mr. Brady" -> Greg;
    "Mr. Brady" -> Peter;
    "Mr. Brady" -> Bobby;
    "Mr. Brady" -> Marcia;
    "Mr. Brady" -> Jan;
    "Mr. Brady" -> Cindy;
    "Mrs. Brady" -> Greg;
    "Mrs. Brady" -> Peter;
    "Mrs. Brady" -> Bobby;
    "Mrs. Brady" -> Marcia;
    "Mrs. Brady" -> Jan;
    "Mrs. Brady" -> Cindy;
}
```

Adding Style

Styling is where your graphs come alive. Colors, shapes, and layout choices turn raw data into a visual story your audience can understand at a glance. This section introduces the tools that make that possible.

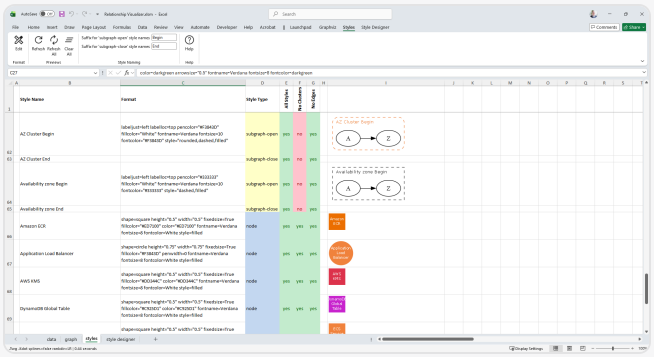
Use the cards below to explore each styling capability in depth.

Style Designer



Define reusable styles for nodes, edges, and clusters.

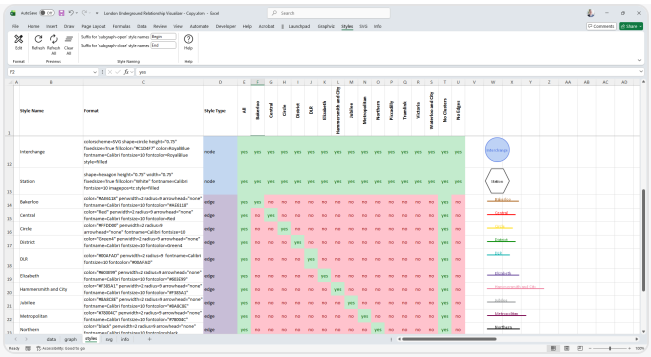
Style Gallery



Save and organize your custom styles for reuse.

Create Views

Apply styles selectively to highlight specific data.



Style Designer

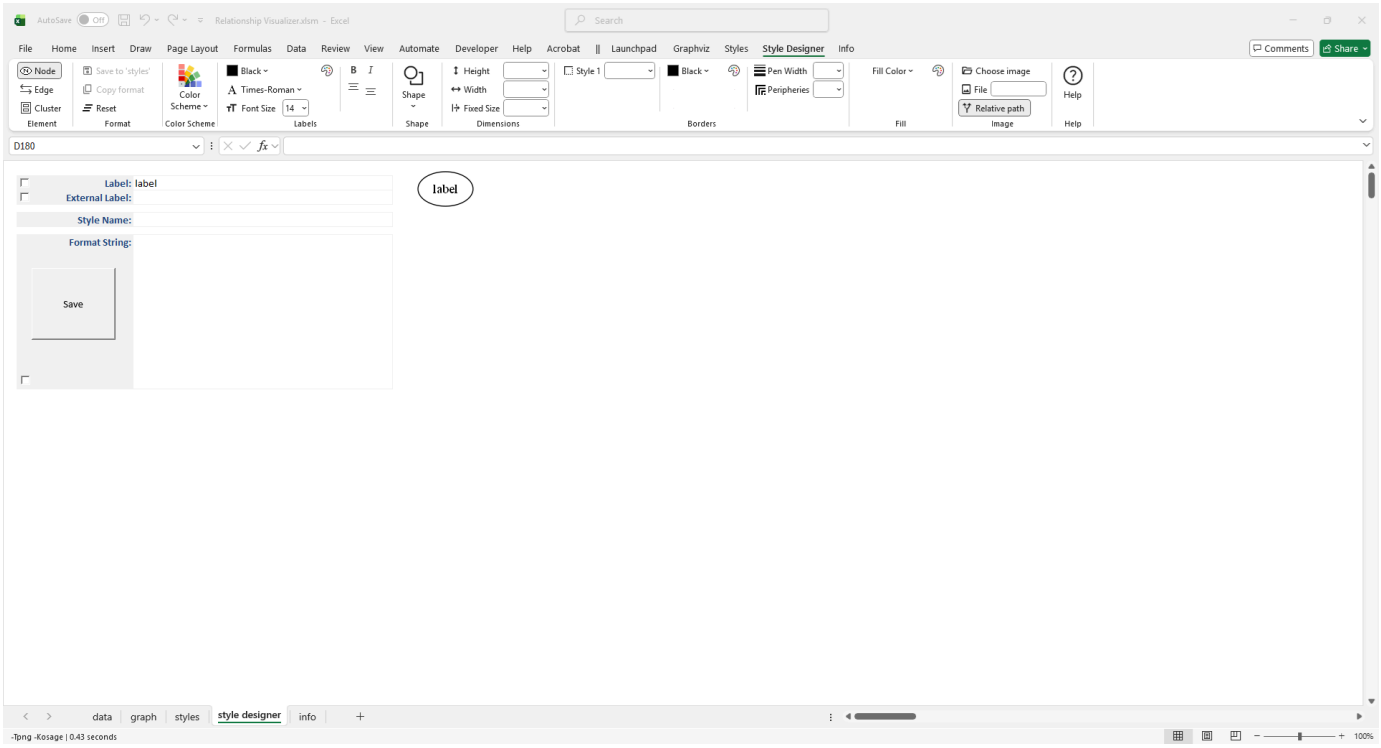
The Graphviz DOT language includes many attributes that control the appearance of nodes and edges. The `style designer` worksheet helps you compose style specifications without needing to know every detail of the DOT language.

Quick Links

Overview

The **Style Designer** worksheet provides an adaptive interface for composing Graphviz style specifications.

The worksheet appears as follows:



It consists of the following constructs:

- **Ribbon Controls** - clickable choices for Graphviz's visible attributes.
- **Label Fields** - preview areas where you can enter and view text.
- **Style Name** - the name assigned to the style definition on the `styles` worksheet
- **Preview Image** - generated by Graphviz to show exactly how your combination of attributes will be rendered.
- **Format String** - the underlying style specification containing the Graphviz attributes.
- **Save Button** - saves the style definition to the **Styles** worksheet, where it can be applied to rows in the **Data** worksheet.

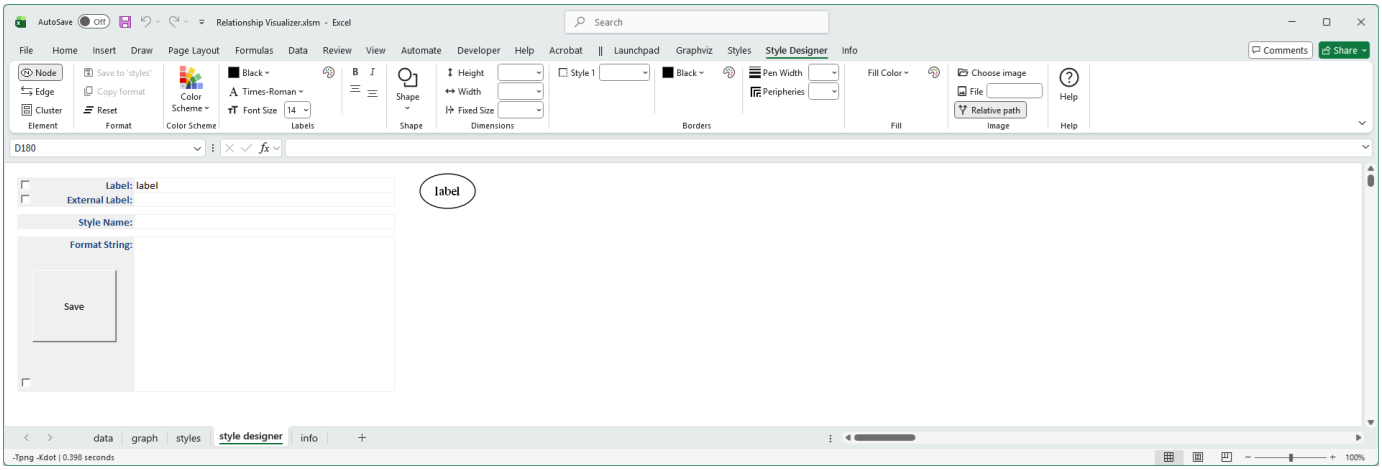
Ribbon Controls

The Style Designer ribbon tab provides three dynamic design modes, controlled by the Element radio buttons in the left-most group. These modes let you create **node styles**, **edge styles**, and **cluster styles**. The ribbon controls update automatically as you make selections.

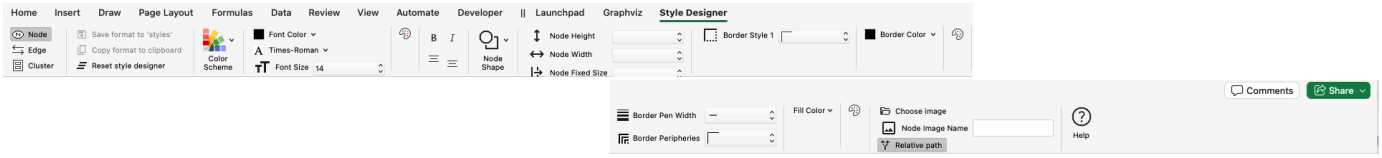
`Node` design mode

Displays the Graphviz node-related attributes.

Windows



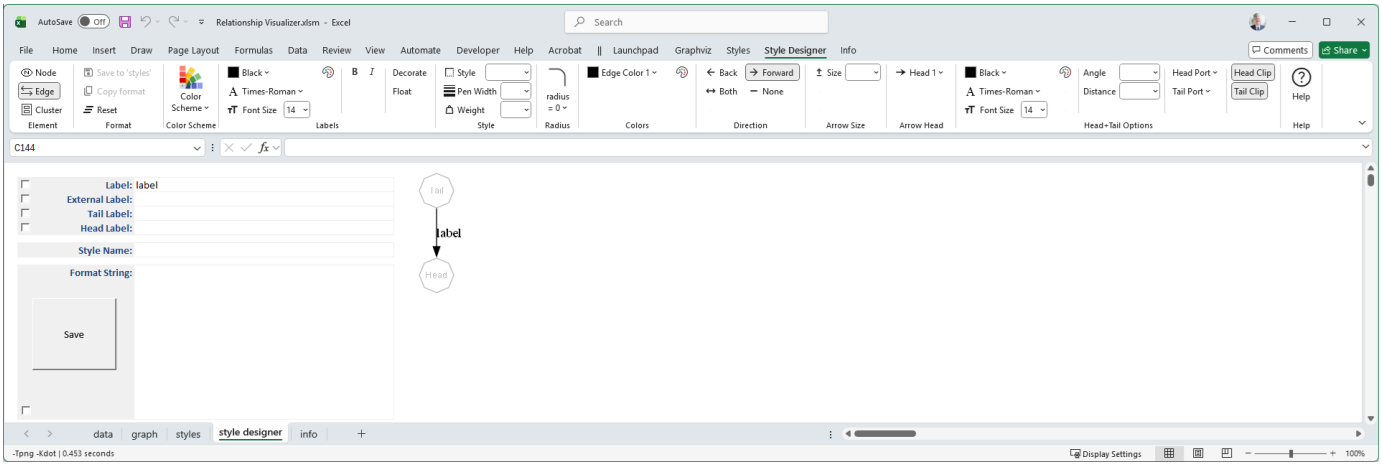
macOS



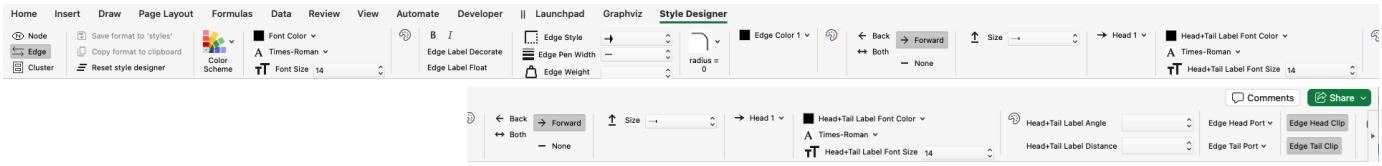
`Edge` design mode

Displays the Graphviz edge-related attributes.

Windows



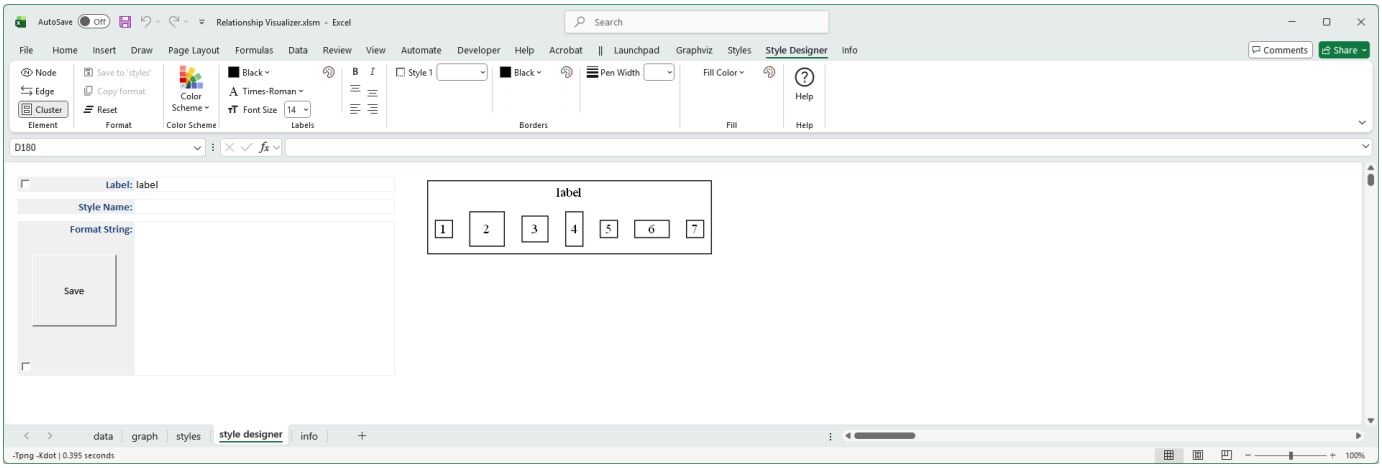
macOS



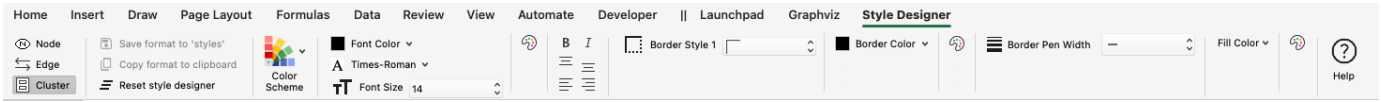
`Cluster` design mode

Displays the Graphviz cluster-related attributes.

Windows



macOS



You define styles by making selections on the **Style Designer** ribbon tab. As you choose options, a format string is generated, and a sample rendering of the node, edge, or cluster is produced using the graphing engine and spline values from the **Graphviz** ribbon tab (explained later).

Use these elements as guides when making selections on the **Style Designer** worksheet, ensuring that appropriate Graphviz attributes are applied in context. For example, when *Element = Edge*, attributes such as `shape` are not offered because they are not valid for edges.

In each design mode, you can experiment with different values until you achieve a visually pleasing result.

Label Fields

The **Label Fields** let you define text that appears in the preview image on nodes, edges, or clusters.

The fields shown depend on the current **design mode** and what Graphviz supports in that context:

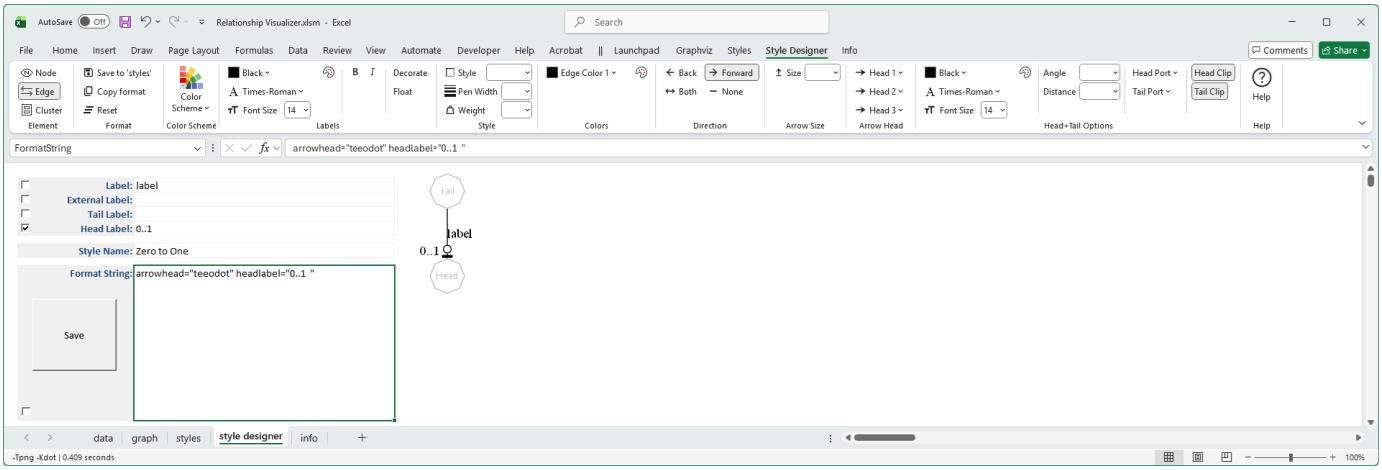
Design Mode	Label	External Label	Tail Label	Head Label
Node	✅	✅		
Edge	✅	✅	✅	✅
Cluster	✅			

Each label field has an associated **check box**:

- **Checked** → The label is included in the style definition and will appear whenever the style is applied.
- **Unchecked** → The label is shown only as representational text in the preview image, not part of the saved style.

Example

Suppose you are defining an edge style to represent a zero-to-one relationship. By entering the caption **“0:1”** in the *Head Label* field and checking its box, the label will be included in the style definition. Whenever this "Zero to One" edge style is used, the “0:1” caption will automatically appear next to the arrowhead.



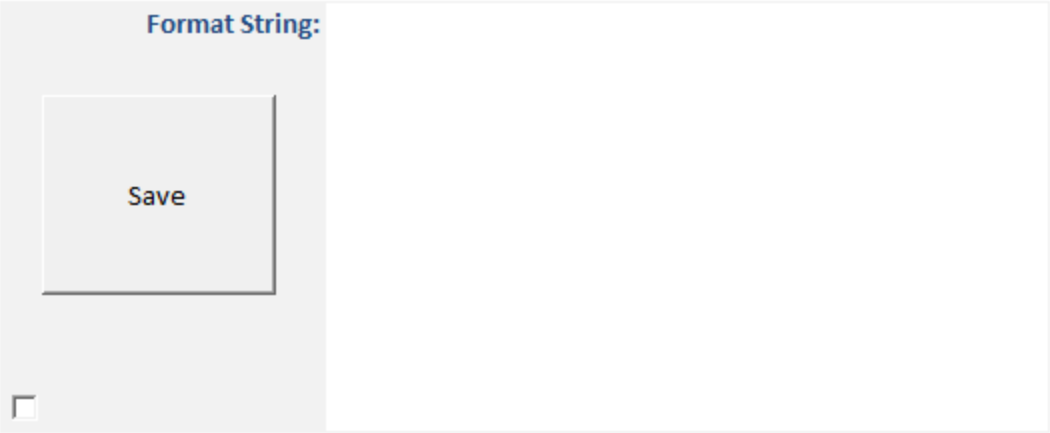
Style Name

This cell contains either:

- The name you want to assign to a **new** style definition.
- The **existing** name of the style definition on the `styles` worksheet which is being modified.

Format String

As you make selections the **Format String** cell builds a list of Graphviz style attributes and writes them to the large cell below:



The **Format String** cell is also an **active cell**, meaning you can edit it directly to fine-tune settings beyond the options provided in the drop-down lists.

For example:

- The font size list jumps from 36 to 48.
- If you want a font size of 40, you may type the value directly into the cell.

⚠️ **Important Notes**

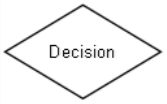
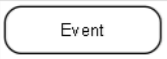
- Any change made in the **Ribbon** will overwrite hand-made edits in the Format String, since ribbon changes rebuild the specification.
- Conversely, deleting **all** the contents of the **Format String** cell will reset the Ribbon settings back to their default values.

Save Button

The large **Save** button, along with the **Save to 'styles'** button in the Ribbon, saves the contents of the **Format String** using the specified **Style Name** on the **Styles** worksheet.

- Each saved style definition is stored as a row in the **Styles** worksheet.
- These saved styles can then be applied to rows in the **Data** worksheet.
- This workflow allows you to build a library of reusable node, edge, or cluster styles.

For example, the image below shows three **Node** style definitions created with the **Style Designer** and saved on the **Styles** worksheet:

Style Name	Format	Style Type	All Styles	No Clusters	No Edges	
Decision	shape=diamond height="0.75" width="1.25" fixedsize=True fillcolor="white" fontname=Arial fontsize=10 style=filled	node	yes	yes	yes	
Event	shape=rectangle height="0.375" width="1.25" fixedsize=True fillcolor="white" fontname=Arial fontsize=10 style="rounded,filled"	node	yes	yes	yes	
Extract	shape=triangle height="0.75" width="1.25" fixedsize=True fillcolor="white" fontname=Arial fontsize=10 style=filled	node	yes	yes	yes	

Color

Graphviz Color Schemes

Color is a key aspect of any visualization, and the **Style Designer** provides full support for all Graphviz color schemes.

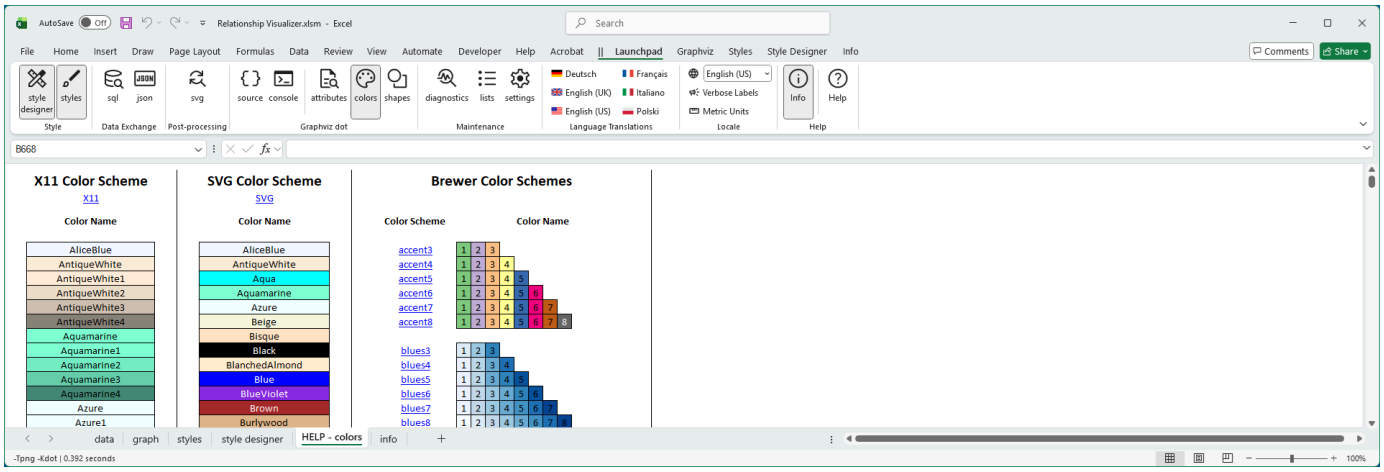
Graphviz defines a *color scheme* as the context for interpreting color names.

If a color value has the form `"xxx"` or `"/xxx"`, then the color `xxx` is evaluated according to the current color scheme. If no color scheme is set, the standard **X11** naming is used.

For example, if `colorscheme="bugn9"`, then `color="7"` is interpreted as `/bugn9/7`.

This may sound complicated, so let's simplify:

- The **Colors** button on the **Launchpad** ribbon shows or hides the **HELP – colors** worksheet.
- This worksheet lists all supported Graphviz color schemes (267 in total).
- Each scheme contains between 3 and 656 colors.



This worksheet is used behind the scenes to generate preview images for color choices.

Style Designer Color Schemes

Graphviz supports multiple color scheme families, which define how color names are interpreted. All Graphviz color schemes are supported in the Style Designer.

The **Style Designer** ribbon provides a large **Color Scheme** button and color drop-down lists to help you select and apply these schemes.

Major Color Scheme Families

X11

- Graphviz's default color scheme.
- Largest predefined set of colors (656 choices).
- Useful for broad, general-purpose visualization with familiar names like *HotPink1* or *LightSkyBlue*.

SVG

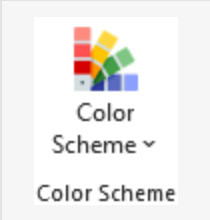
- Matches the standard color set defined by the SVG specification.
- Smaller, web-friendly palette of named colors.
- Ideal for consistency with web graphics and cross-platform rendering.

Brewer

- Based on Cynthia Brewer's *ColorBrewer* palettes, designed for data visualization.
- Provides carefully balanced sequential, diverging, and qualitative color schemes.
- Useful for maps, charts, and diagrams where perceptual uniformity and accessibility are important.

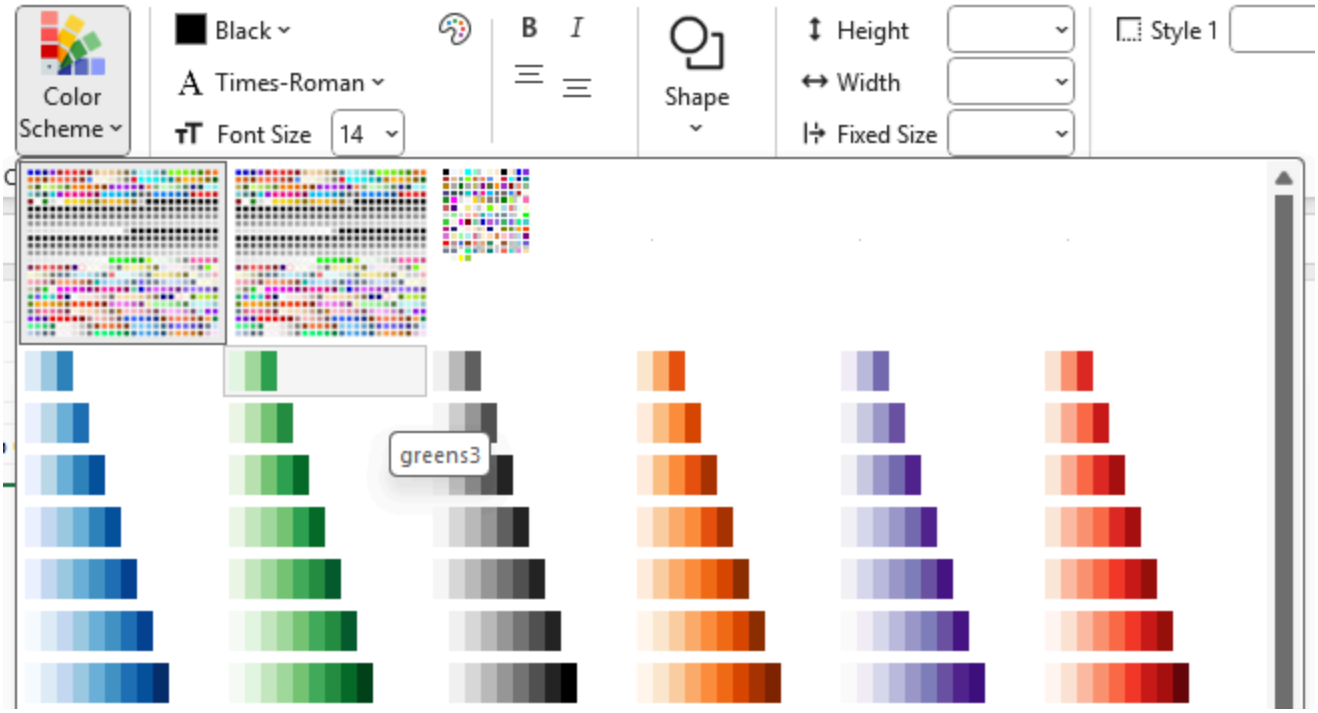
Choosing a Color Scheme

Clicking the **Color Scheme** button opens a gallery where you can choose a scheme:

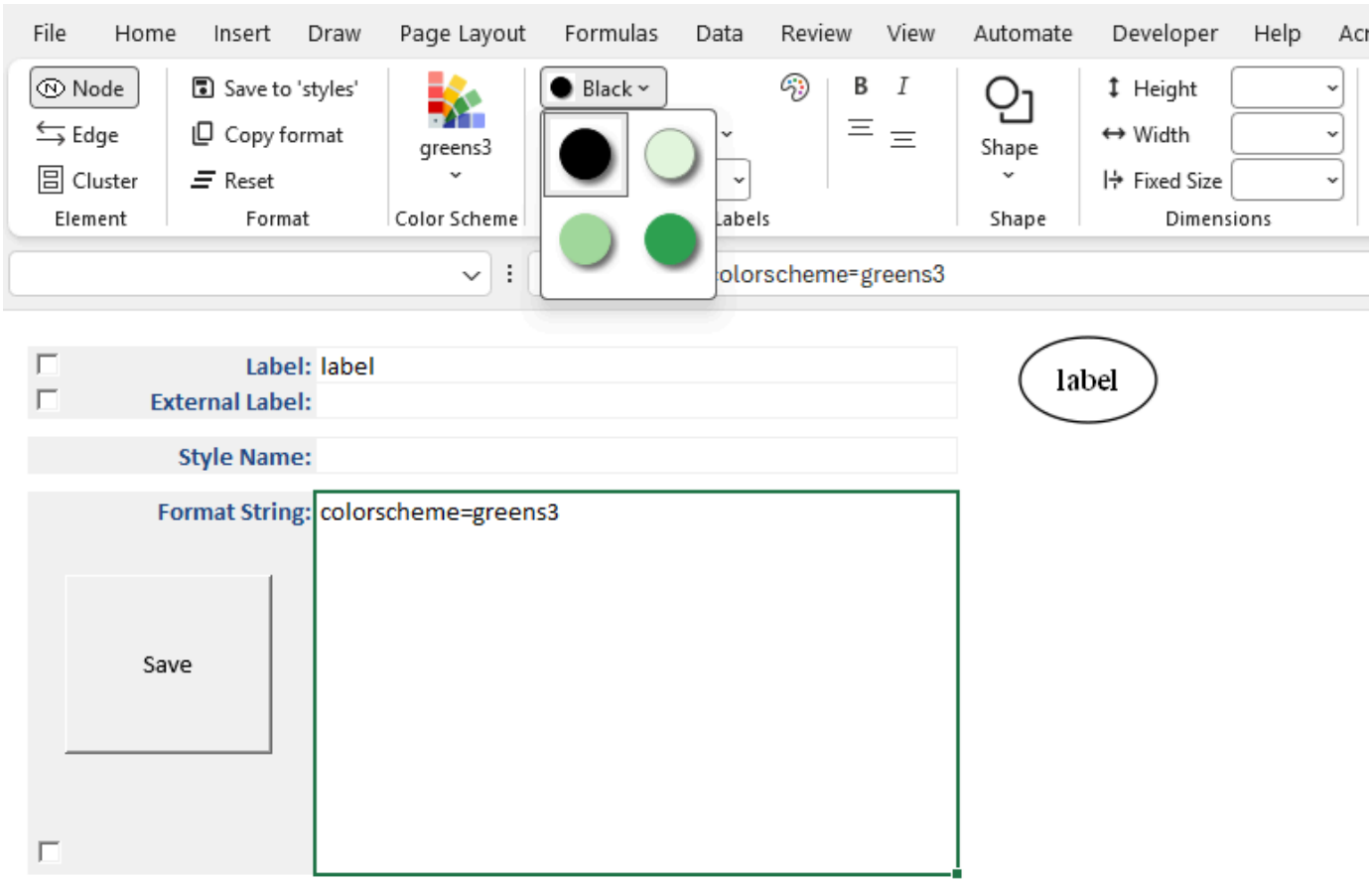


When you select a scheme, all color-related drop-down lists are refreshed to display the colors for that scheme.

For example, choosing `greens3`

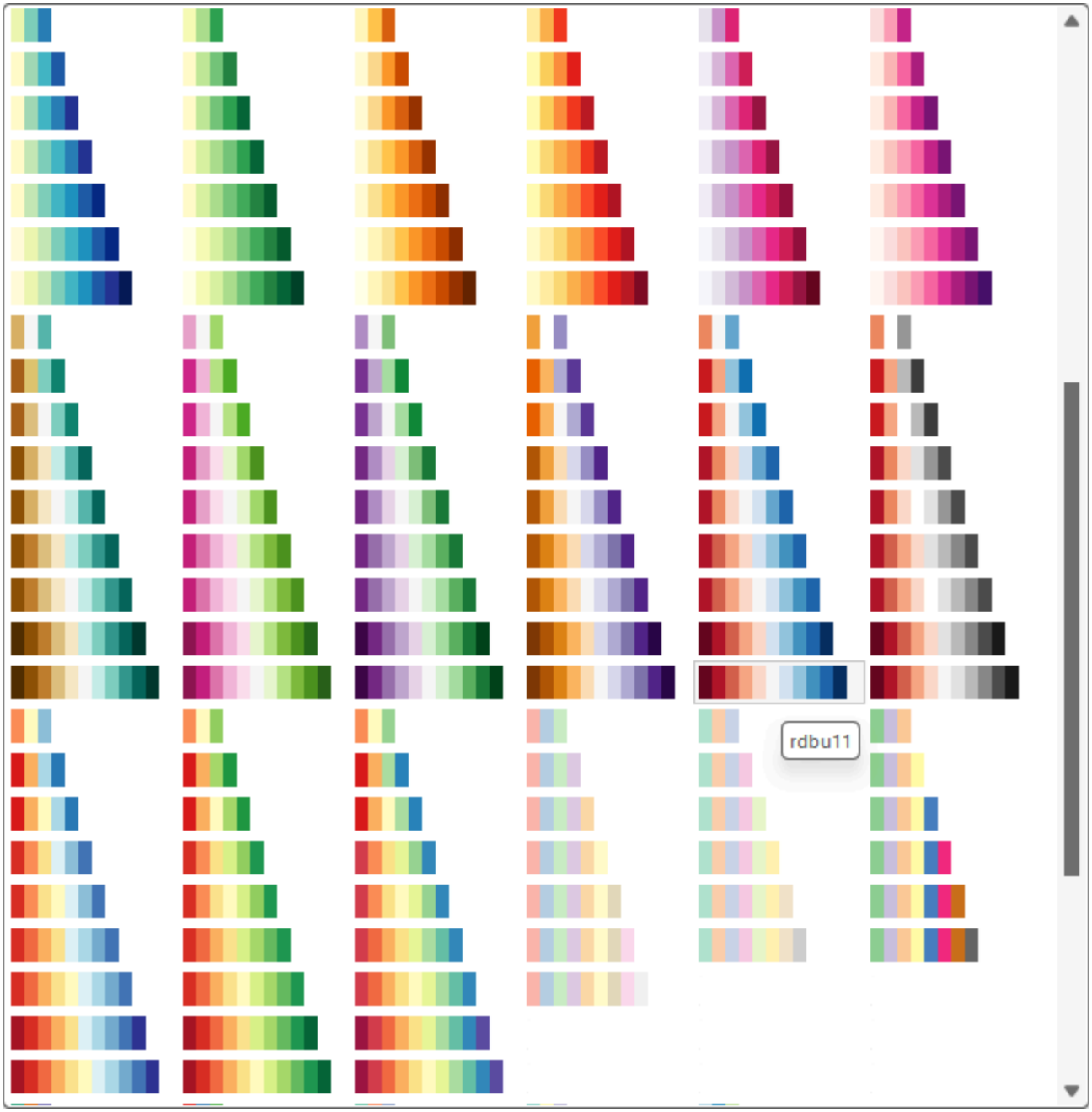


updates the lists to values `1`, `2`, and `3`, displays color icons, and adds the attribute `colorscheme=greens3` to the **Format String**.

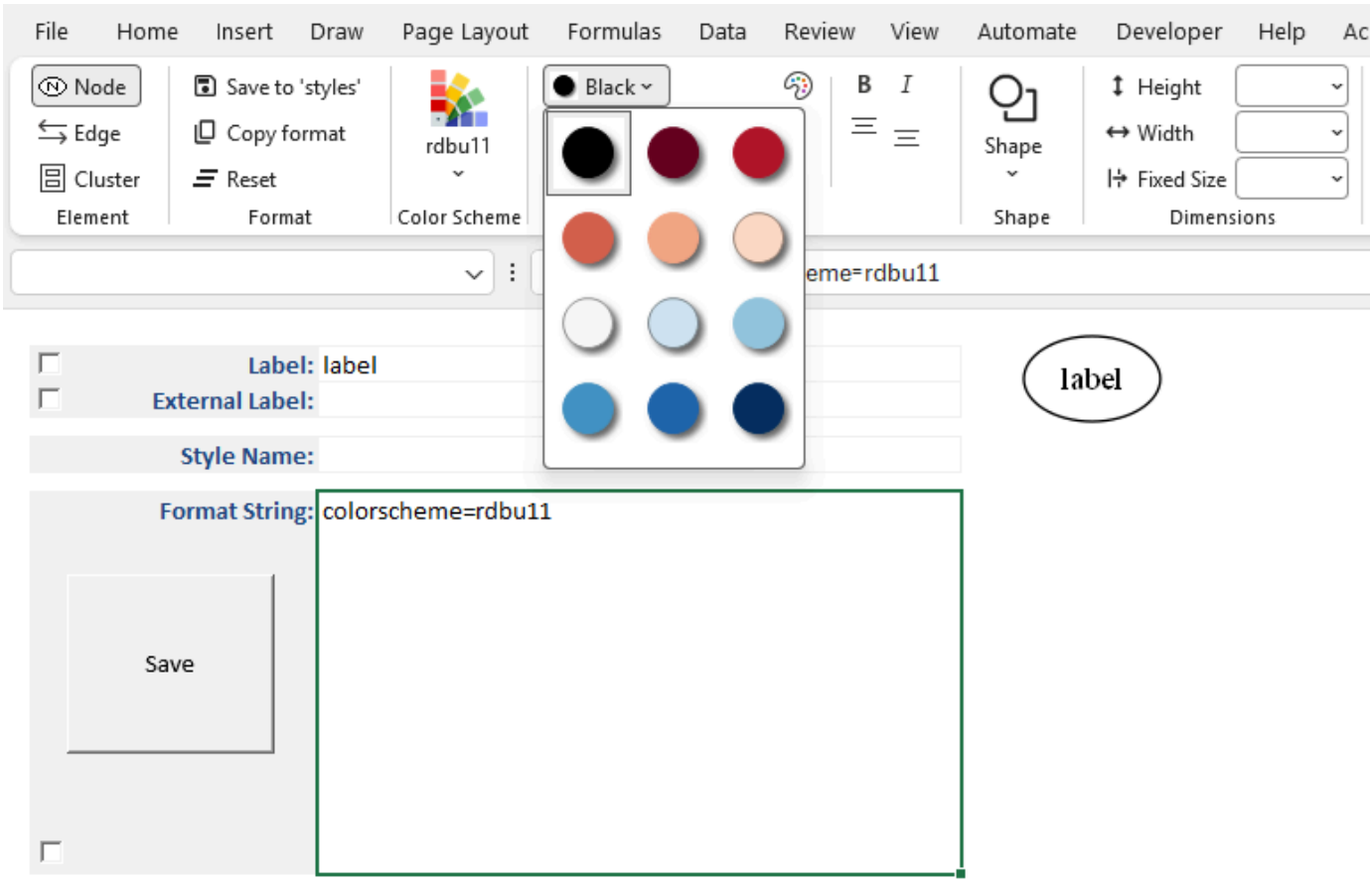


If you switch to another scheme, the lists refresh again.

For example, after selecting `greens3`,



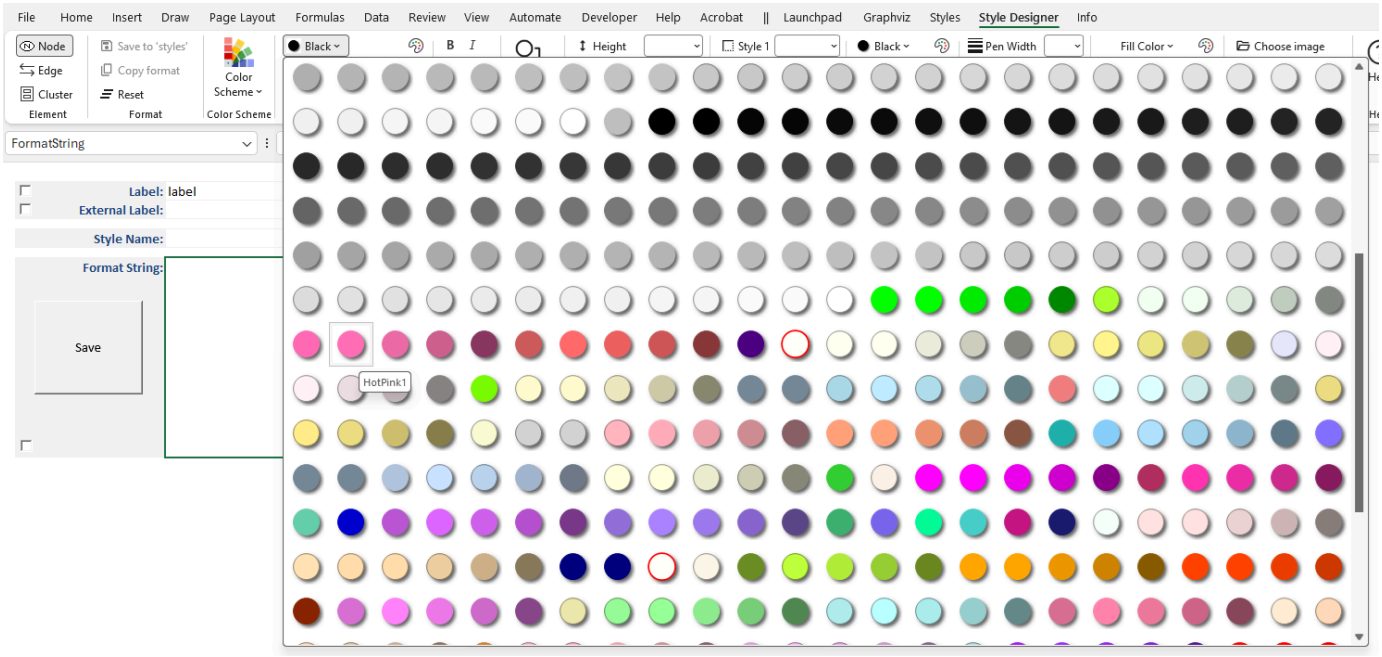
choosing `rdbu11` updates the lists to values `1` through `11`, displays color icons, and adds `colorscheme=rdbu11` to the **Format String**.



Selecting a Pre-defined Color

You choose colors by clicking on any of the color drop-down arrows to open a gallery of available colors.

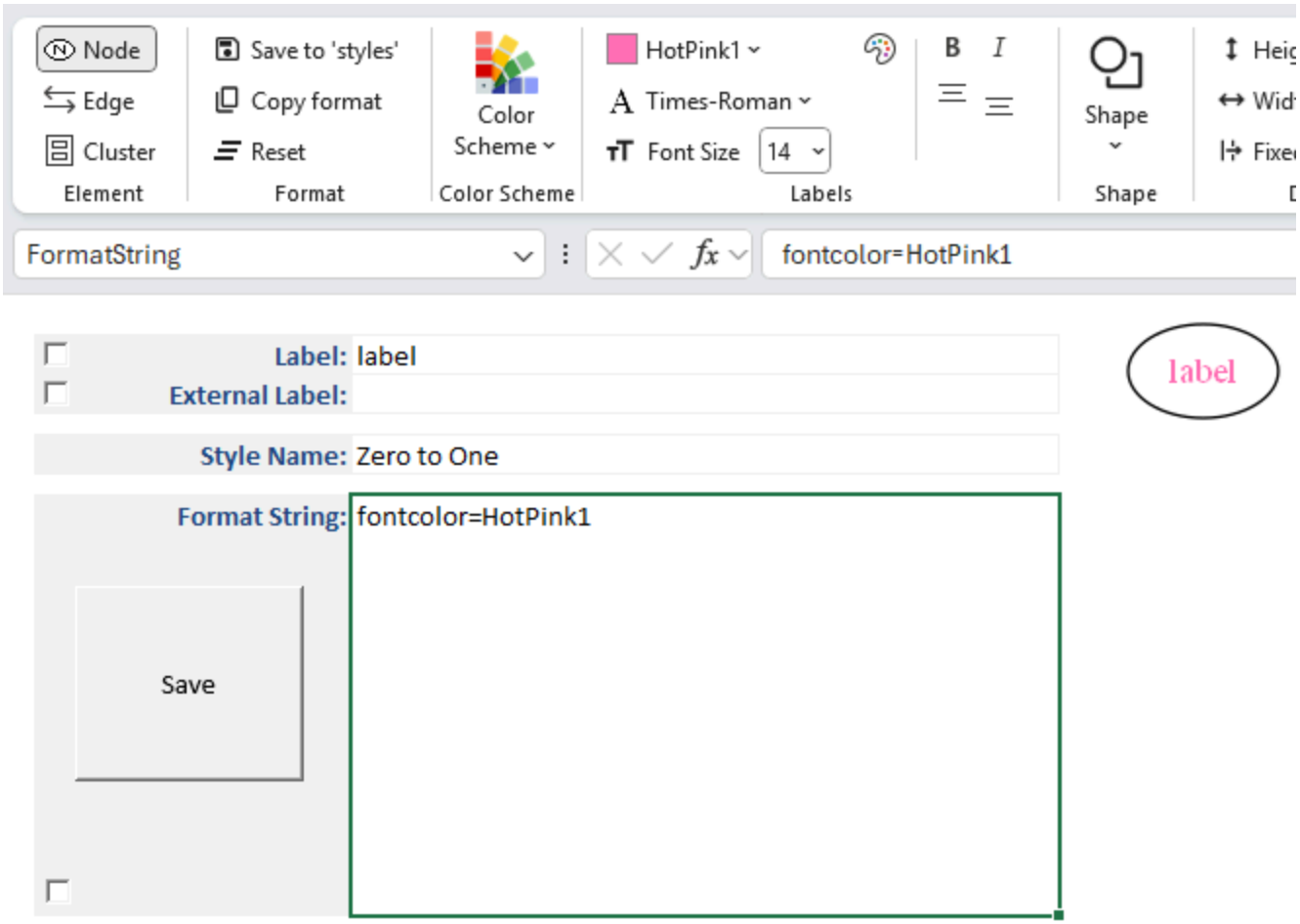
Hovering over a color shows its name, such as *HotPink1* in the example below:



When you click on a color, the **Style Designer** ribbon updates to show both the color and its name.

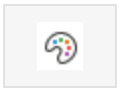
The color name is added as an attribute in the **Format String**, and a preview image is generated to show how the color will appear when rendered by Graphviz.

In the example below, `HotPink1` has been selected as the font color:

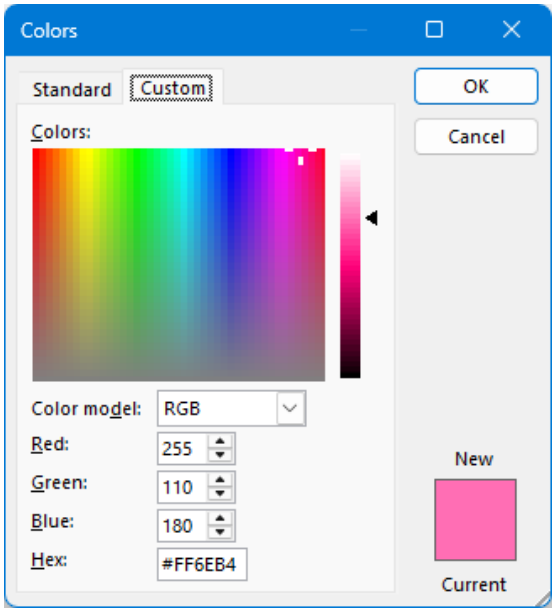
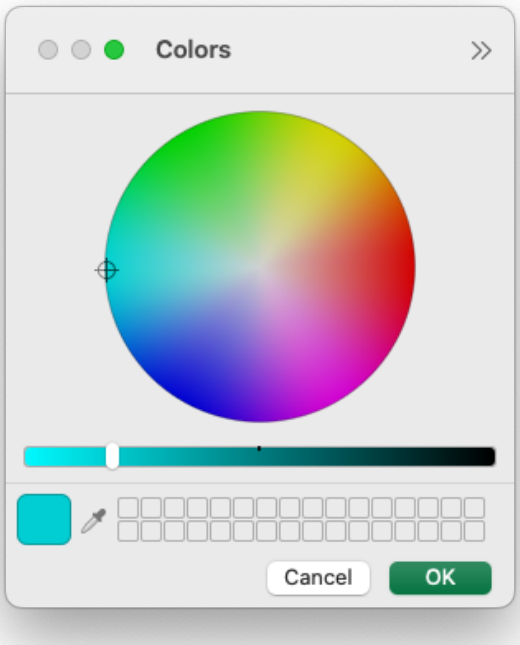


Selecting a Color Using RGB (Red Green Blue) Values

In addition to choosing from predefined color schemes, you can specify a custom color using the **Color Dialog**. To the right of each color choice dropdown is a small button with a color icon which appears as:



This dialog provides a native interface for selecting colors on your operating system:

Windows 11 Color Dialog	macOS Color Dialog
	

The **Color Dialog** is initialized to the currently chosen color.

If a named color from a color scheme is selected, it is automatically converted to RGB when the Color Dialog is displayed.

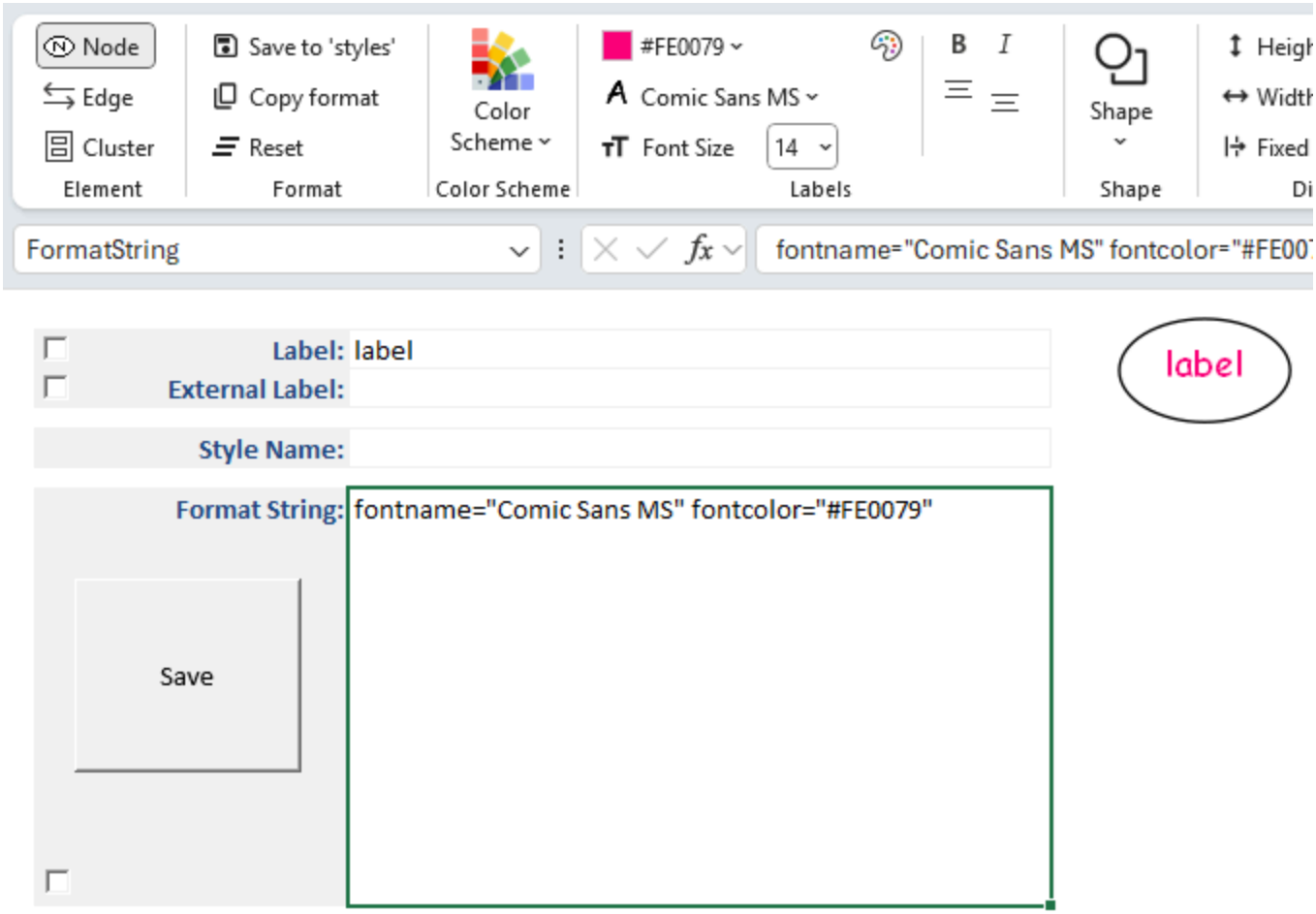
This allows you to first choose a color from a scheme, then refine it as needed using the picker.

For example, you might use the dialog to set a font color to a specific RGB value rather than relying on scheme-based names.

When you select a color in the dialog:

- The chosen color is displayed as an icon in the **Style Designer** ribbon, and the RGB value of the color is shown as the color name (for example, `#FE0079`).
- The color name or RGB value is added as an attribute in the **Format String**.
- A preview image is generated to show how the color will appear when rendered by Graphviz.

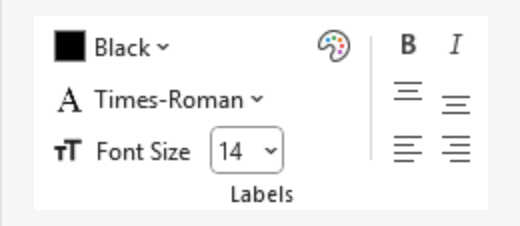
In the example below, both the **Font Name** and the **Font Color** have been specified, with the font color defined as an RGB value:



Labels

You can design styles which format label text using the following controls:

- Color
- Font
- Font size
- Bold
- Italic
- Label Location



Label Fonts

The **Style Designer** font drop-downs present a gallery of fonts that Graphviz can render on your chosen operating system:

- **Windows** → The list is derived from the fonts installed on your PC, filtered to remove fonts known to be incompatible with Graphviz.
- **macOS** → A static list of fonts is provided from the **lists** worksheet.

When the **Font Name** drop-down is selected for the first time, Graphviz generates a preview image of the letters **Aa Bb Cc** for each font.

These preview images are cached for future use. You may notice a slight delay the first time as the cache is built, but subsequent displays occur quickly.

An example **Font Name** gallery on Windows 11 appears as follows:

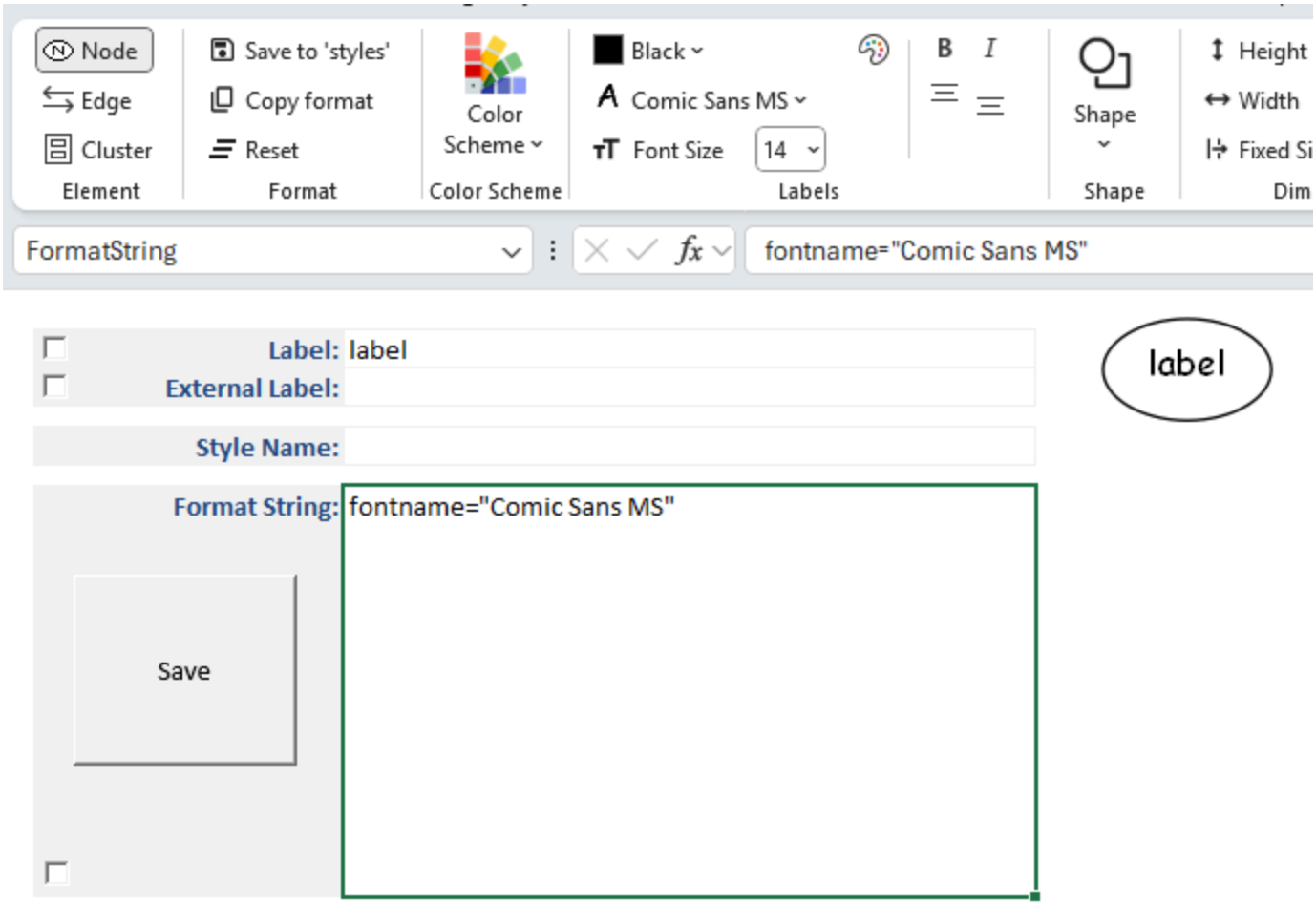
Selecting a Font

The currently selected font name is highlighted in the gallery.

When you choose a font (e.g., `Comic Sans MS`):

- The **Font Name** caption on the drop-down changes to the selected font.
- An icon of the letter **A** in the font appears in the ribbon to the left of the **Font Name**.
- The font name is added as an attribute in the **Format String**.
- A new preview image is generated, showing the associated labels rendered in the chosen font.

For example:



Label Location

Text can be aligned relative to the borders of a shape or cluster. Alignment is available as follows via the alignment buttons:

Position	Node	Cluster
Top	✅	✅
Center	✅	✅

Position	Node	Cluster
Bottom	✅	✅
Left		✅
Middle		✅
Right		✅

Shapes

Graphviz provides a wide variety of **node shapes** that you can apply in the **Style Designer**. Shapes define the overall outline of a node and help visually distinguish different types of elements in your diagram.

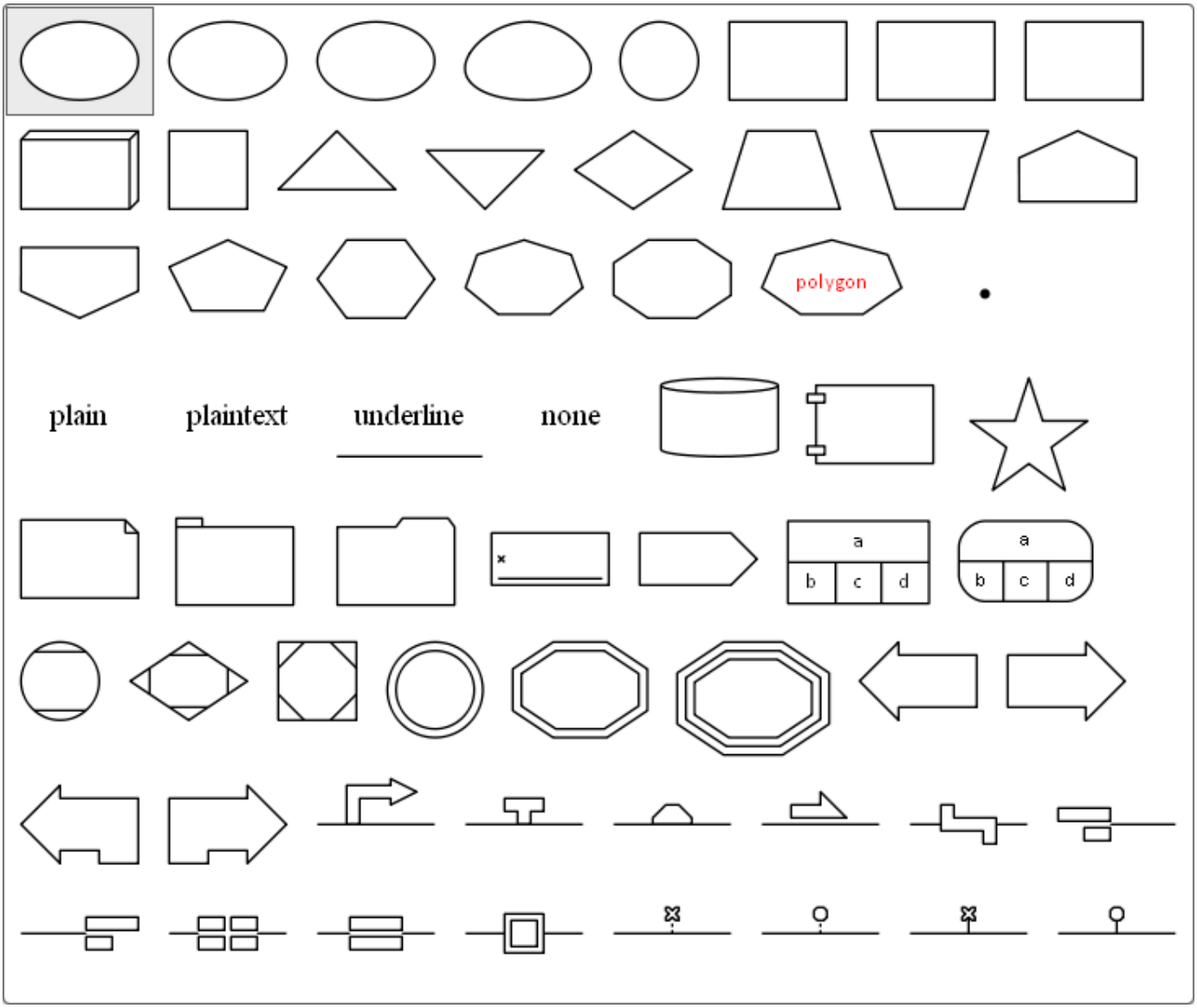
Shapes can be used to convey meaning, organize information, or simply improve the readability of your graph.

For example, rectangles may represent processes, ellipses may represent entities, and diamonds may represent decisions.

Specifying a shape

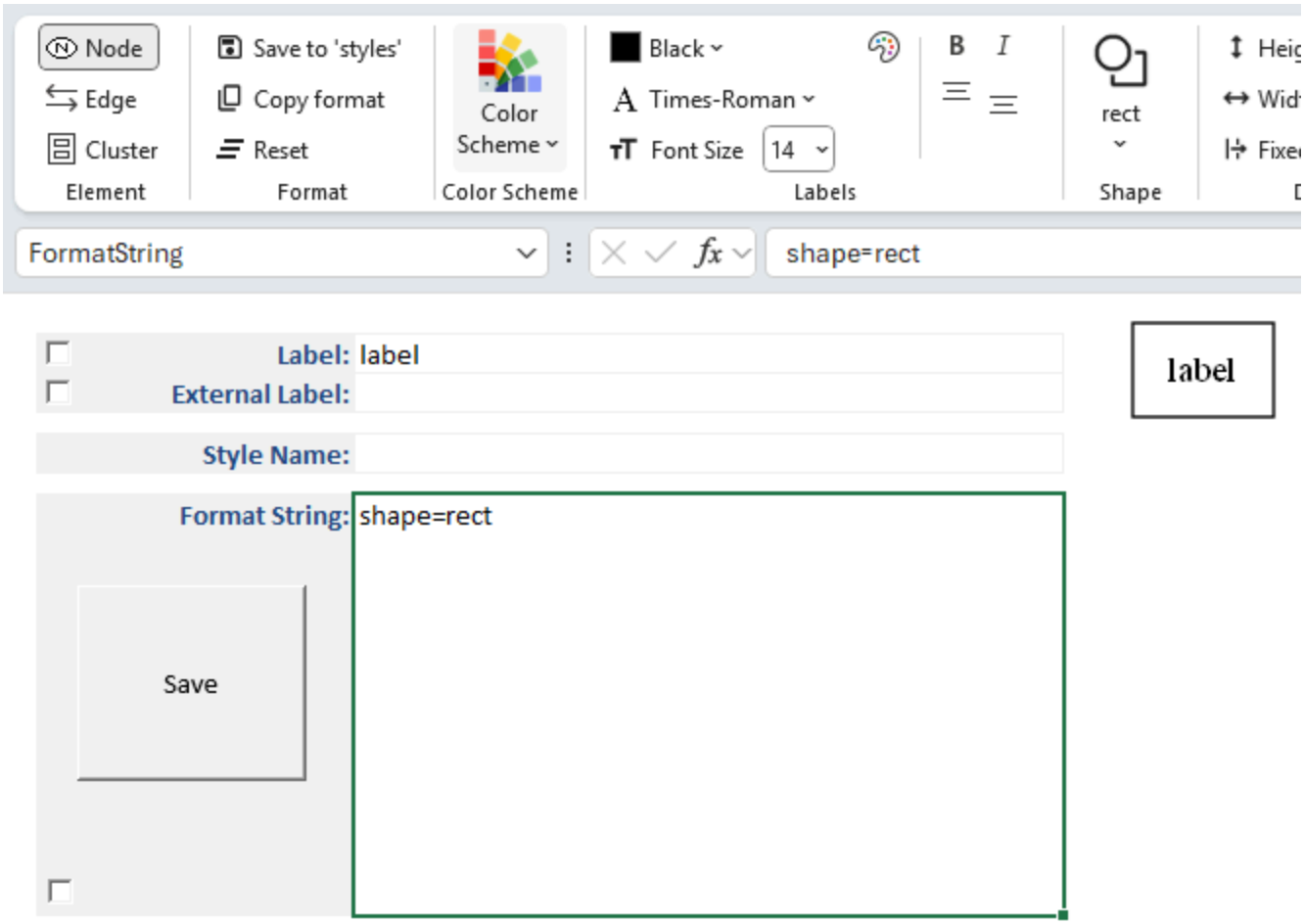
Click on the `Shape` drop-down button.

A gallery of shapes supported by Graphviz is presented showing a sample image of the shape.



Here we pick one of the rectangle shapes. When you select a shape:

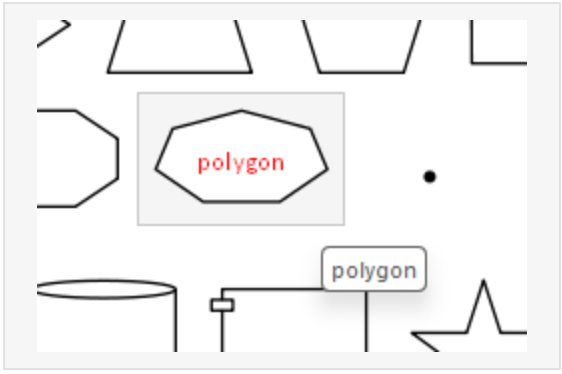
- The name of the chosen shape is displayed in the **Style Designer** ribbon as the caption of the `Shape` button.
- The shape name is added as an attribute in the **Format String** (e.g., `shape=rect`).
- A preview image is generated to show how the node will appear when rendered by Graphviz.



Polygon Shapes

Polygon shapes are unique from other shapes in Graphviz and have extra attributes which control how the polygon is created.

If you select `polygon` as the shape the ribbon will change dynamically to present additional choices as shown below:

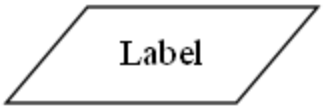


Selecting `polygon` changes the ribbon to appear as:

Polygon Skew

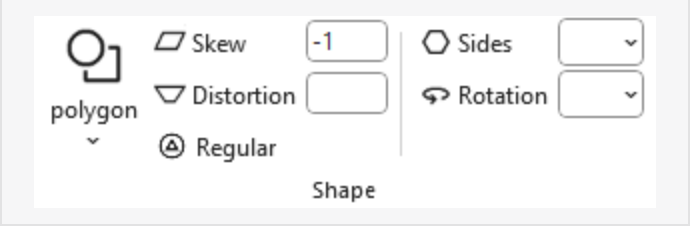
Positive values skew top of polygon to right; negative values skew the top of the polygon to the left.

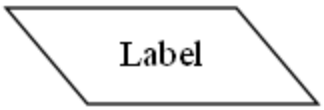
Positive Skew




```
shape="polygon" skew="1"
```

Negative Skew



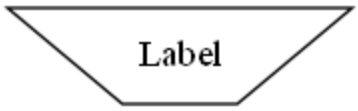


```
shape="polygon" skew="-1"
```

Polygon Distortion

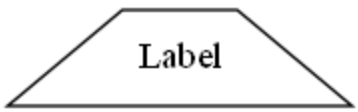
Positive values cause top part of the polygon to be larger than bottom; negative values do the opposite.

Positive Distortion



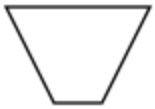

```
shape="polygon" distortion="1" regular="No"
```

Negative Distortion



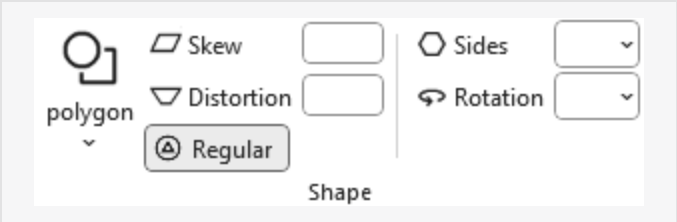

```
shape="polygon" distortion="-1" regular="No"
```

Combining Skew with Distortion

+	skew="-1"	skew="0"	skew="1"
distortion="1"			
distortion="0"			
distortion="-1"			

Regular Polygon

If true, forces the polygon to be regular, i.e., the vertices of the polygon will lie on a circle whose center is the center of the node.




```
shape="polygon" regular="Yes"
```

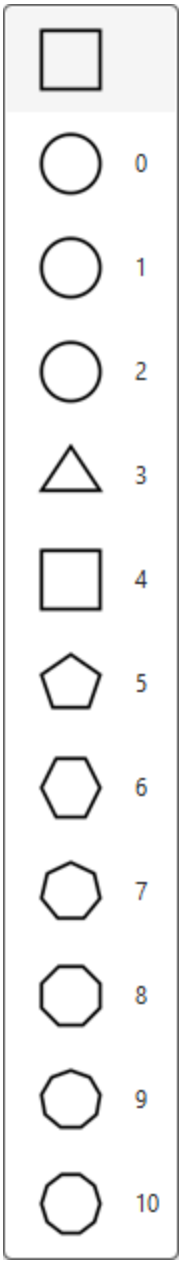
Polygon Sides

The **sides** attribute controls the number of polygon sides used when drawing a node shape.

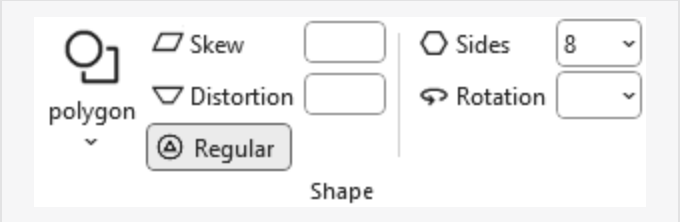
- **Default**: A polygon has **4 sides** (a square).

- **sides < 4**:
  - If the polygon is **not regular**, Graphviz substitutes an **ellipse**.
  - If the polygon is **regular**, Graphviz substitutes a **circle**.
- **sides ≥ 4**:
  - The node is drawn as a polygon with the specified number of sides.
  - For example, `sides=5` produces a pentagon, `sides=8` a hexagon, and so on.

When you set **sides**, the chosen value is displayed in the **Style Designer** ribbon, added to the **Format String** (e.g., `sides=6`), and shown in the preview image.



sides=8




```
shape="polygon" sides="8" regular="yes"
```

Ellipses/circles can also be skewed and distorted to create unique shapes.

sides=1, with skew and distortion


```
shape=polygon sides=1 skew=1 distortion="-1" regular=no
```

Polygon Rotation

The **orientation** attribute controls the rotation angle of a node shape.
It determines how the shape is drawn relative to its default position.

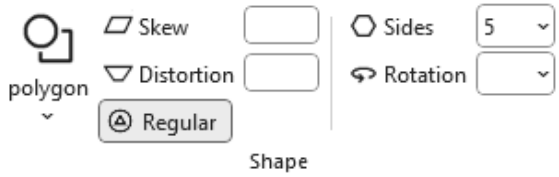
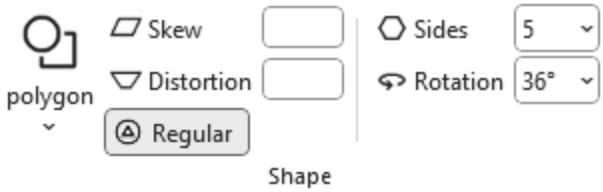
- **orientation=0** (default)

  - The shape is drawn in its standard upright position.

- **orientation=n**
  - The shape is rotated by *n* degrees, **clockwise**.
  - For example, `orientation=45` tilts the shape diagonally, while `orientation=90` rotates it a quarter turn.

- **interaction with regular polygons**
  - When used with polygon shapes (via the **sides** attribute), orientation rotates the polygon around its center.
  - For any number of polygon sides, 0 degrees rotation results in a flat base.
  - This is useful for aligning triangles, diamonds, or other polygons to match the desired layout.

When you set **orientation**, the chosen value is displayed in the **Style Designer** ribbon, added to the **Format String** (e.g., `orientation=90`), and shown in the preview image.

5-sided regular polygon with no rotation	5-sided regular polygon rotated 36 degrees clockwise
	
	

Dimensions

In Graphviz, you can control the **height** and **width** of node shapes to adjust their overall size. These attributes ensure that shapes are scaled consistently and remain readable in your diagram.

Shape Height and Width

When you specify a shape's dimensions:

- The chosen values are displayed in the **Style Designer** ribbon.
- The attributes `height` and `width` are added to the **Format String** (e.g., `height=1.0, width=2.0`).
- A preview image is generated to show how the resized shape will appear when rendered by Graphviz.

By default, Graphviz calculates shape dimensions automatically based upon the computed size of the label and its placement.

Specifying explicit values allows you to emphasize certain nodes, align shapes visually, or ensure uniform sizing across your diagram.

Units of Measure

Graphviz's default unit of measure for shape dimensions is **inches**.

However, you can specify metric units by enabling the **Metric Units** checkbox on the **Launchpad** ribbon.

- When metric units are selected, display values are shown in **millimeters (mm)**.
- These values are automatically converted to **inches** internally to satisfy Graphviz's requirements.
- This allows you to work in familiar metric units while ensuring compatibility with Graphviz's rendering engine.

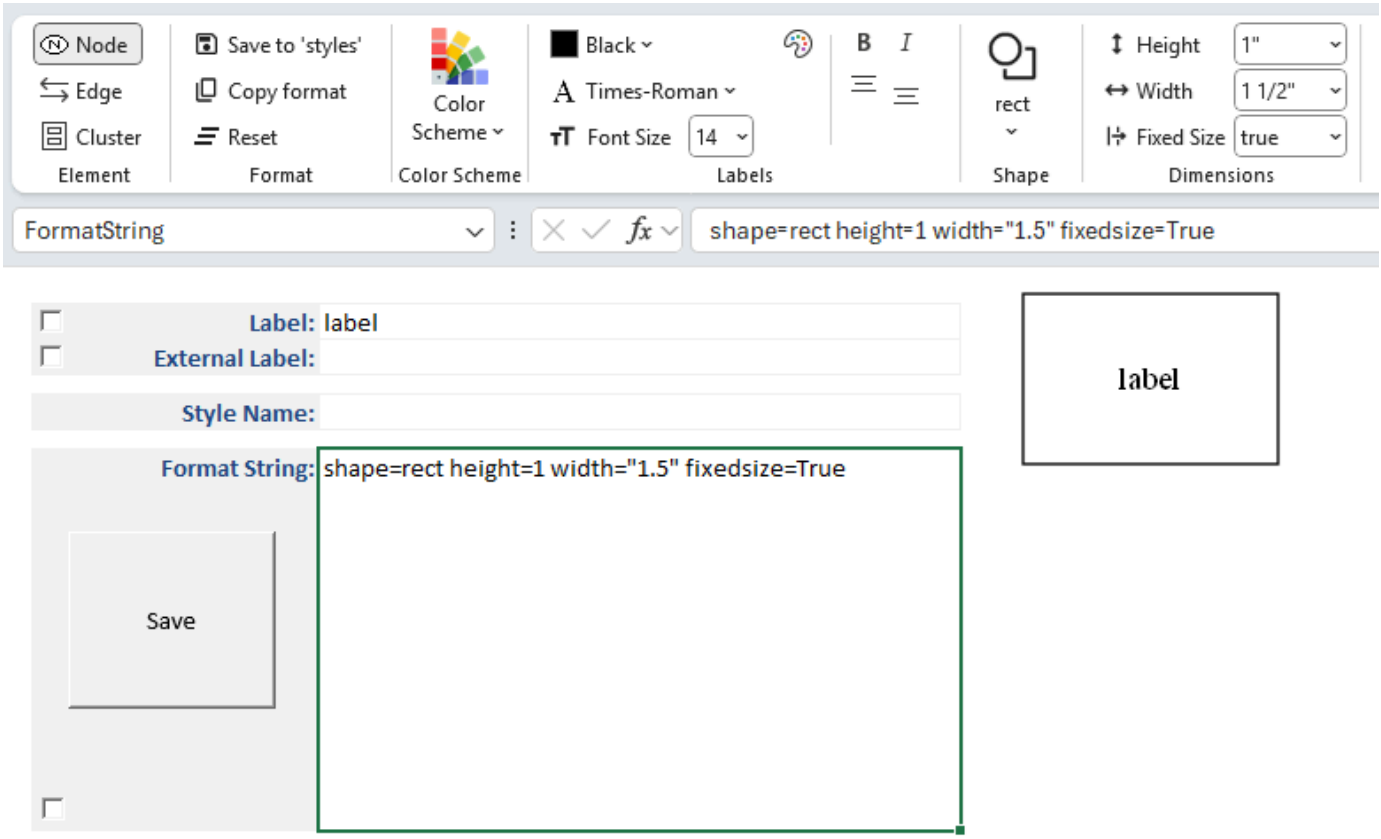
Fixed Size

The **fixedsize** attribute controls whether a node's shape is drawn at a fixed size or allowed to expand to fit its label text.

- **fixedsize=false** (default)
  - The node's dimensions are adjusted automatically to fit the label.
  - The attributes `height` and `width` act as minimum values.
  - Longer labels will stretch the shape horizontally or vertically as needed.
- **fixedsize=true**
  - The node's dimensions are locked to the specified `height` and `width`.
  - Labels which exceed the width of the shape are truncated down equally from the left and right sides to fit inside the fixed shape.
  - Useful for ensuring uniform node sizes across a diagram, regardless of label length.
- **fixedsize=shape**
  - The node's height and width are locked, but the label is allowed to stretch horizontally.
  - This ensures consistent vertical sizing while accommodating longer text.
  - Useful for diagrams where uniform height is desired, but labels vary in length.

When you enable **fixedsize**, the chosen values are displayed in the **Style Designer** ribbon, added to the **Format String** (e.g., `shape=rect height=1 width="1.5" fixedsize=True`), and

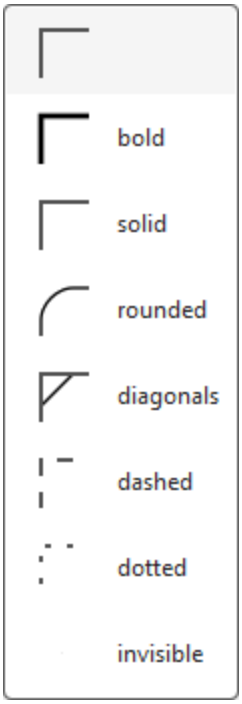
shown in the preview image.



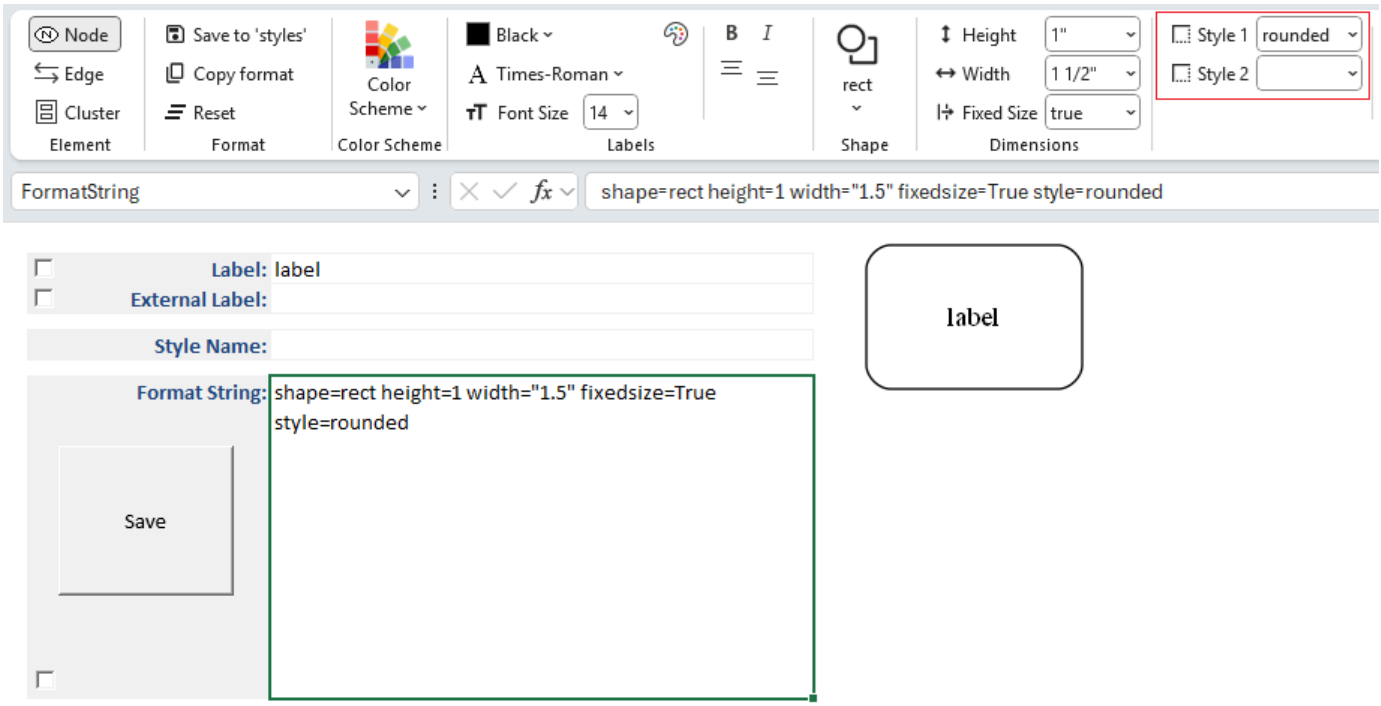
Borders

Border Styles

Up to 3 border styles are selectable and are additive, making it possible to have styles such as bold edge and rounded corners. When you click on any of the **Border Style** drop-down lists you will be presented with the list of choices along with a sample image of the style.



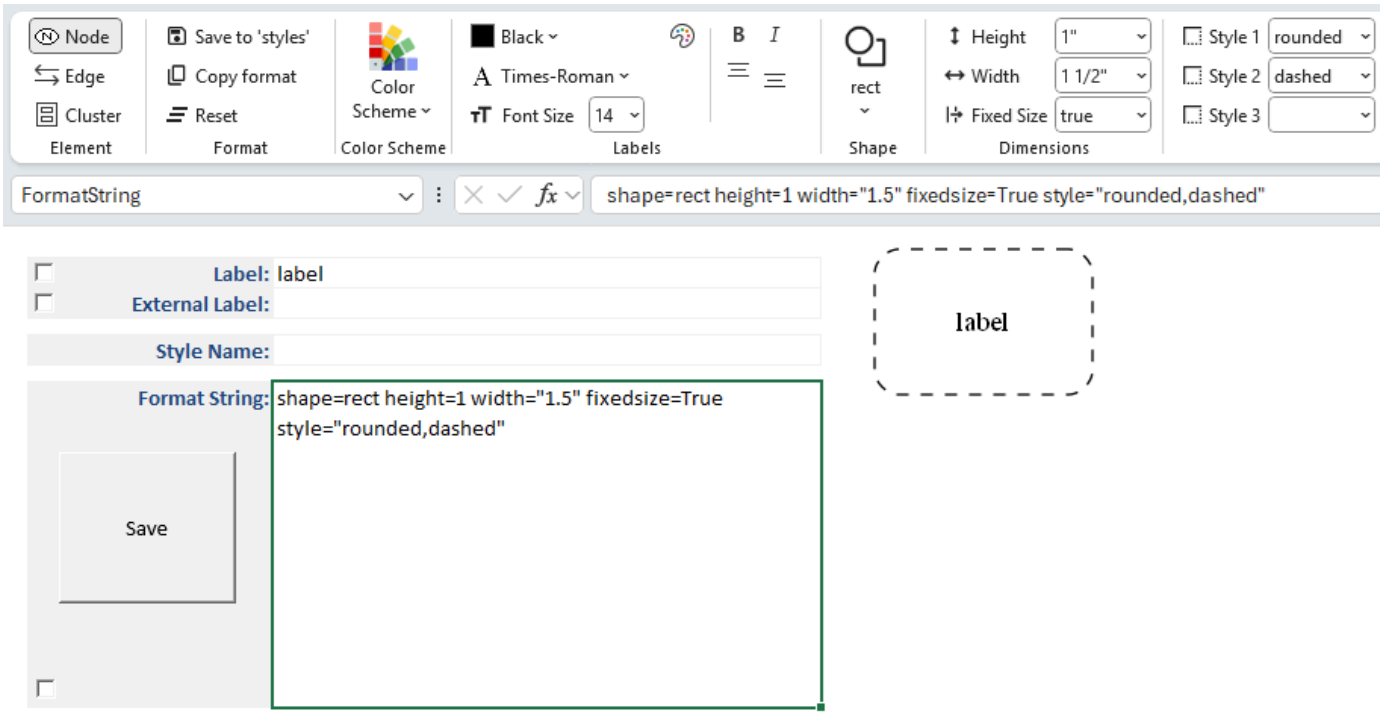
In this example `Style 1` is set to `rounded` to give the rectangle rounded corners.



The **Style Designer** provides an adaptive interface for applying multiple border styles. As you make selections, additional style options appear dynamically:

- Once a style is chosen, the **Style 2** drop-down becomes available.
- Selecting `dashed` as the second choice results in a rounded, dashed border.
- A **Style 3** drop-down then appears, allowing you to continue layering styles.

This adaptive behavior makes it easy to combine multiple visual effects without cluttering the interface.

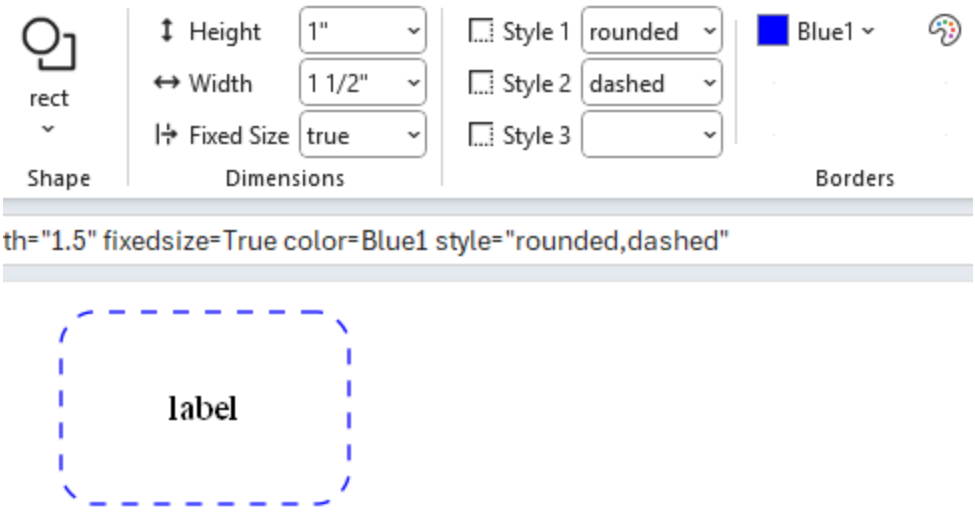


Border Color

How to choose colors has already been explained.

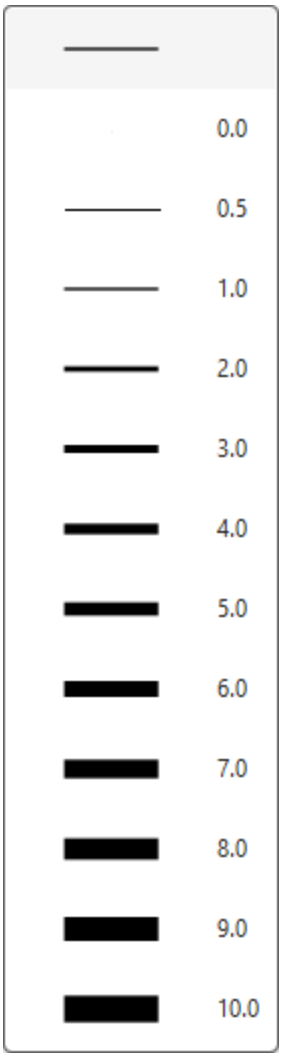
The **Border Color** controls allow you to specify the color of a node shape or cluster border.

For example:



Border Pen Width

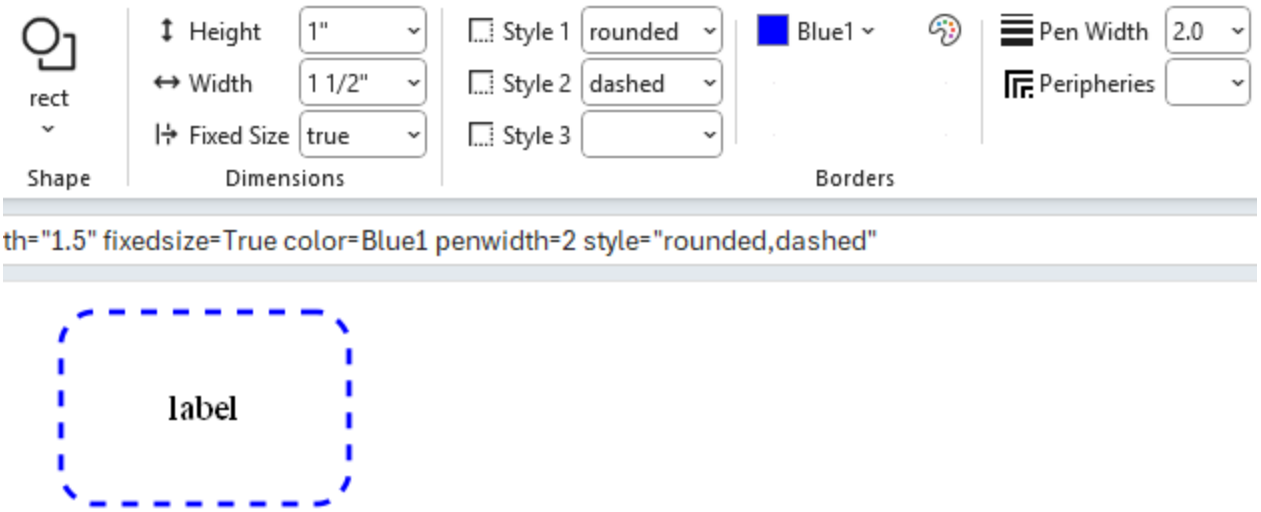
The **penwidth** attribute controls the thickness of lines used to draw node borders and edges.



- **penwidth=1.0** (default)
  - Standard line thickness.
  - Borders and edges are drawn with a single-pixel width.
- **penwidth>1.0**
  - Increases line thickness proportionally.
  - For example, `penwidth=2.0` doubles the thickness, while `penwidth=3.0` triples it.
  - Useful for emphasizing certain nodes or edges in a diagram.

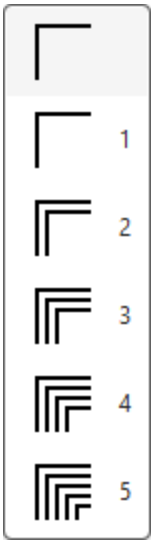
- **penwidth<1.0**
  - Decreases line thickness.
  - For example, `penwidth=0.5` produces a thinner line than the default.
  - Can be used for subtle or secondary connections.

For example:



Border Peripheries

The **peripheries** attribute controls how many borders (or outlines) are drawn around a node shape.



- **peripheries=1** (default)
  - A single border is drawn around the shape.
- **peripheries=2**
  - Two concentric borders are drawn, giving the node a “double-outlined” appearance.
- **peripheries=n**
  - Any positive integer value `n` draws that many concentric borders.
  - Useful for visually emphasizing certain nodes or distinguishing categories.

For example:


```
th="1.5" fixedsize=True color=Blue1 penwidth=2 peripheries=2 style="rounded,dashed"
```

Fills

Fill Color

The **fillcolor** attribute controls the interior color of a node shape or cluster. It determines how the inside of the shape is rendered, providing contrast with the border and improving visual clarity.

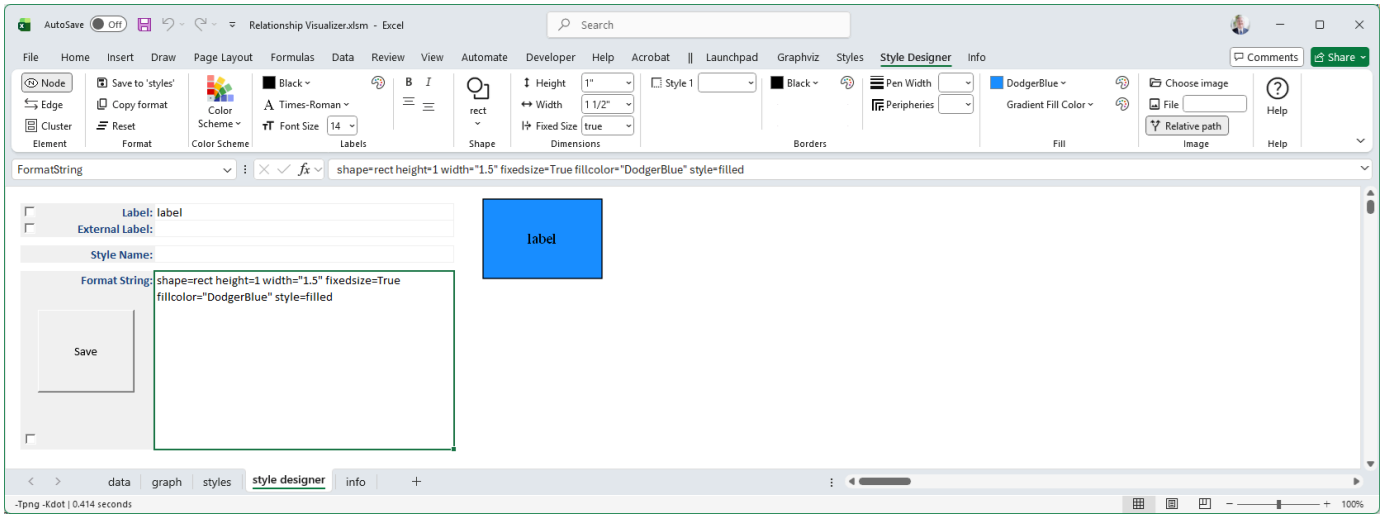
Fill colors can be selected from predefined color schemes or refined using the **Color Dialog**.

They are often used to group related nodes, highlight important elements, or improve the overall readability of a diagram.

When you specify a fill color:

- The chosen color is displayed in the **Style Designer** ribbon, and the `Fill Color` caption changes to the name or RGB value of the color chosen.
- A new dropdown for `Gradient Fill Color` appears below the fill color.
- The color name or RGB value is added as an attribute in the **Format String** (e.g., `fillcolor=DodgerBlue`).
- A preview image is generated to show how the node will appear when rendered by Graphviz.

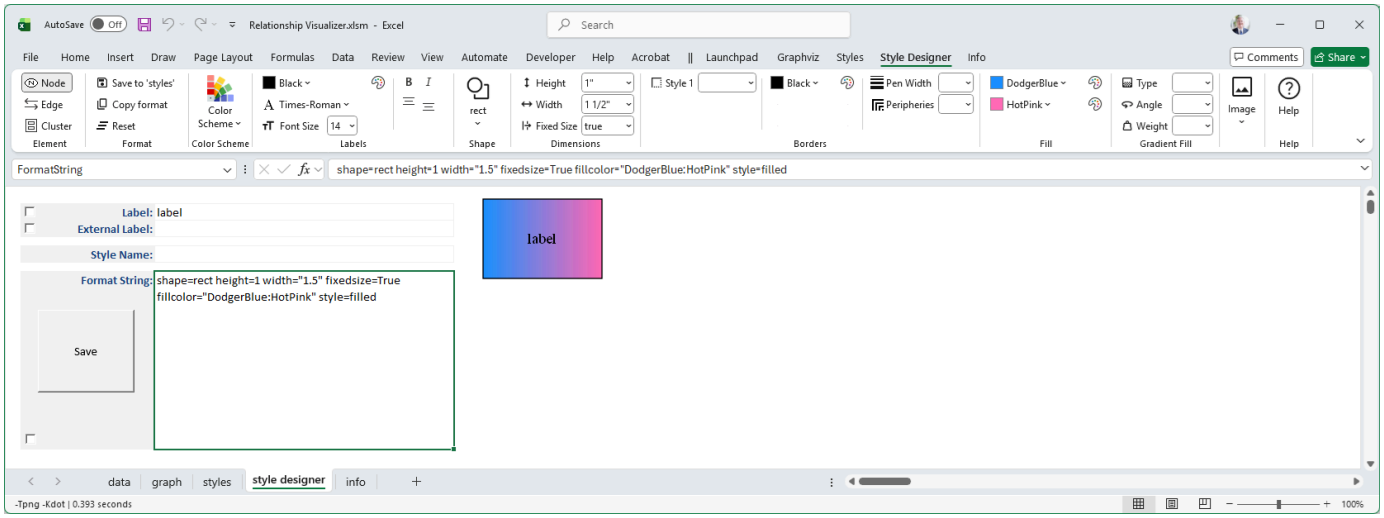
For example:



Gradient Fill Color

Notice that the ribbon dynamically changes once a `Fill Color` is specified to display a new choice for `Gradient Fill Color`.

A `Gradient Fill Color` allows you to select a second color which the Fill Color will gradually transition to. If you select `HotPink` as the `Gradient Fill Color` the preview image changes to look like:



Another set of dynamic changes occur as three additional choices, **Type**, **Angle**, and **Weight** appear to the right of the fill color selections. These choices allow you to define how the gradient transition occurs.

Gradient Type

The **gradienttype** attribute controls how multiple colors blend together inside a node shape or cluster. Instead of filling the shape with a single solid color, you can apply a gradient to create smooth transitions between colors.

Gradient types can be used to highlight relationships, emphasize categories, or simply add visual appeal. For example, a vertical gradient may suggest progression, while a radial gradient can emphasize a central point.

The Gradient Type is either `filled` (i.e., linear) or `radial`.

The differences are illustrated below:

`gradienttype=filled`	`gradienttype=radial`
	

Gradient Angle

The **gradientangle** attribute controls the direction of a gradient fill inside a node shape or cluster.
It determines how the colors specified in the **fillcolor** attribute are blended across the shape.

Changing the Gradient Angle moves the angle of the gradient fill.

- For linear fills, the colors transform along a line specified by the angle and the center of the object.
- For radial fills, a value of zero causes the colors to transform radially from the center; for non-zero values, the colors transform from a point near the object's periphery as specified by the value.

Angles are measured in **degrees**, with `0` representing a left-to-right horizontal gradient.
Other values rotate the gradient clockwise:

- `90` → top-to-bottom vertical gradient
- `180` → right-to-left horizontal gradient
- `270` → bottom-to-top vertical gradient

By adjusting the gradient angle, you can control the visual flow of color transitions, highlight directionality, or add subtle emphasis to your diagram.

For example, if you change the Gradient Angle to 90 degrees, the preview images now appear as:

`gradienttype=filled gradientangle=90`	`gradienttype=radial gradientangle=90`
	

Gradient Weight

The **gradient weight** is specified as part of the **fillcolor** attribute, not as a separate attribute. It controls how much influence each color has in a gradient fill.

When you define a gradient:

- The colors are listed in the **fillcolor** string, separated by a colon.
- A semicolon followed by a numeric value (between 0.0 and 1.0) specifies the weight.
- The weight determines the balance between the first and second colors.

For example, specifying a gradient weight of 20% for the fill color is specified as:

```
style=filled fillcolor="DodgerBlue;0.20:HotPink"
```

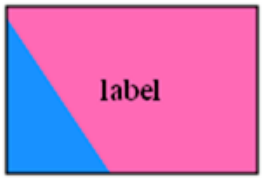
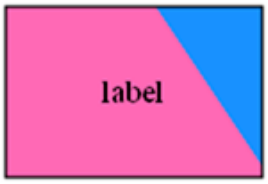
and the image appears as:

By adjusting the gradient weight, you can highlight one color more strongly, create subtle shading effects, or achieve balanced transitions between multiple colors.

Gradient Weight + Gradient Angle

Gradient Angle can be combined with the Gradient Weight to rotate the position of the color split, as in these examples where the gradient weight of the blue fillcolor is 20%:

`gradientangle=0`	`gradientangle=90`	`gradientangle=180`	`gradientangle=270`
			

`gradientangle=45`	`gradientangle=135`	`gradientangle=225`	`gradientangle=315`
			

Images

As you develop more advanced relationship graphs you may want to use images to represent the nodes in combination with, or in place of the node shapes. Graphviz supports an `image=` attribute where you can provide a file name of an image to include in a node.

The Relationship Visualizer by default will look for images in the directory where the spreadsheet is saved.

Image Storage and Paths

If you wish to store images in other locations, you must either:

- Make a configuration change on the **Settings** worksheet to specify the location(s).
  - The image path must be defined before you can use the `image=` attribute in a style definition.
- Include the path to the image directly, in either **Relative** or **Absolute** form.

Relative vs. Absolute Paths

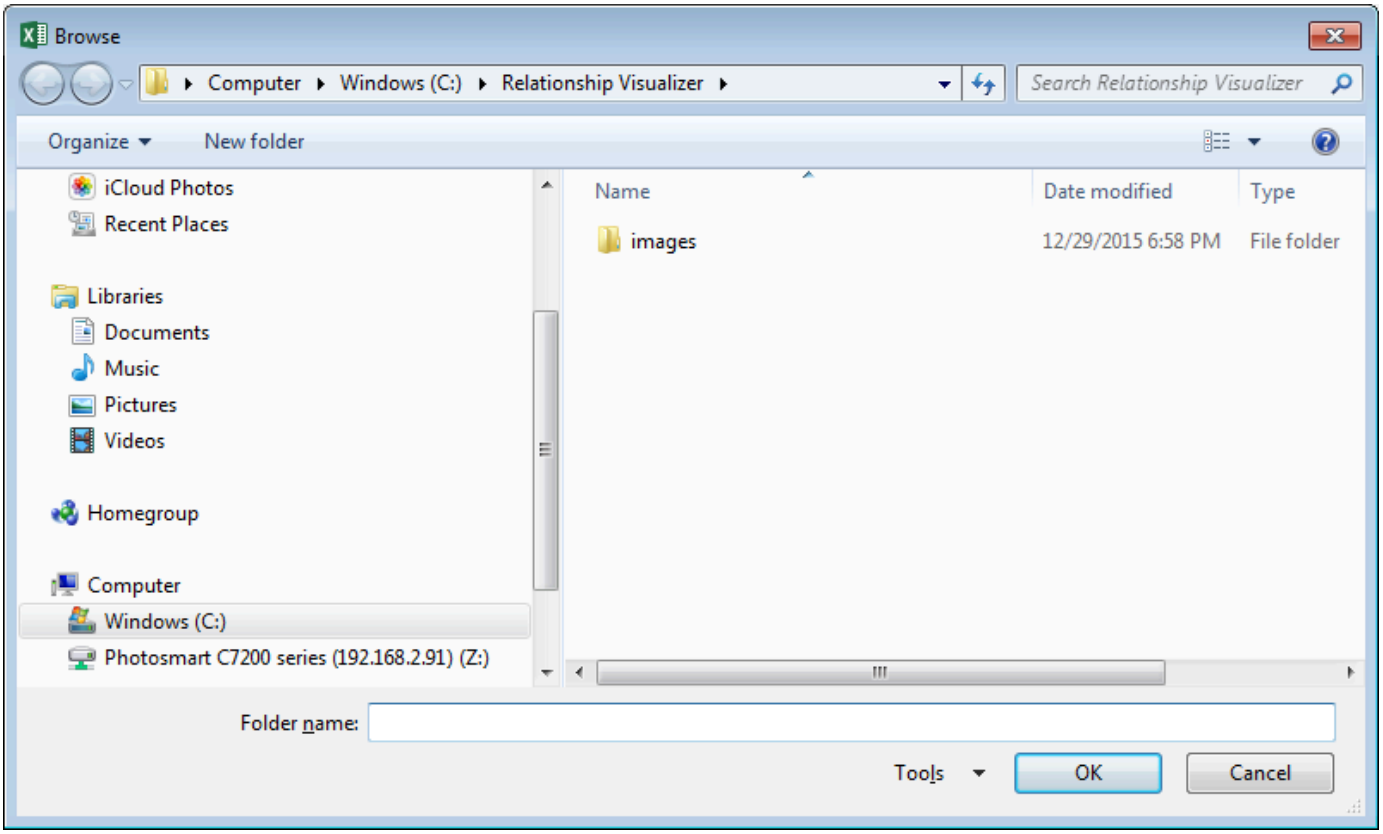
- **Relative Path**
  - Specifies the image location relative to the workbook directory.
  - Example: `images/logo.png`
  - ✅ Easier portability. If the workbook and images are kept together in a folder, cloning or moving the folder preserves the links automatically.
  - ✅ Ideal for sharing with others or using across multiple devices.
- **Absolute Path**
  - Specifies the full location of the image on your system.

- Example: `C:/Users/Jeffrey/Documents/Graphviz/images/logo.png`
- ✅ Ensures the image is always found, regardless of where the workbook is located.
- ✅ Useful when images are stored in a central repository or shared network drive.
- ⚠️ Less portable. Moving the workbook without the same directory structure will break the link.

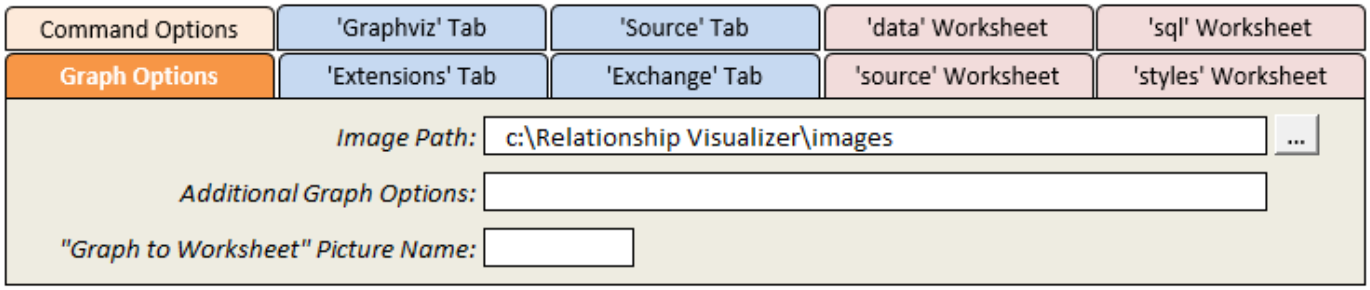
By choosing the appropriate path type, you can balance **portability** (relative paths) with **certainty of location** (absolute paths).

Add an image path

Switch to the `settings` worksheet and locate the "Image Path:" setting in the **Graph Options** section. To the right of the cell is a button with three dots [...]. If you press that button it will bring up the standard directory selection dialog which you can use to choose the directory where the images are stored. Navigate to the directory and press the "OK" button to transfer the path to the cell.



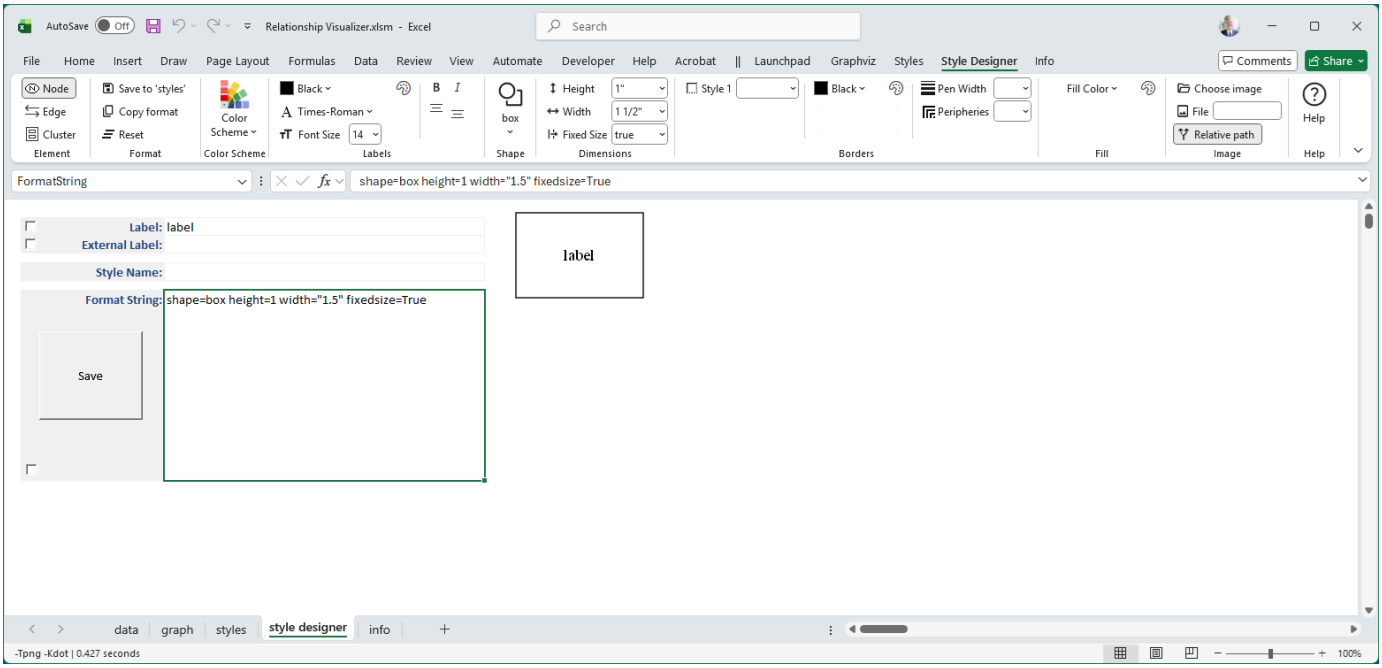
Your settings should appear like this:



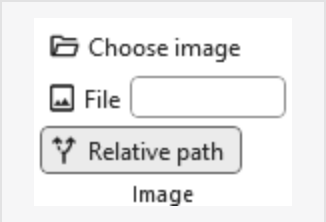
Specify an image

Image name is an option on the `style designer` worksheet that is useful when you want to create a common style definition where all nodes of a given style use a common icon. For example, it is possible to depict computers with one image, depict databases with another image, and depict computer programmers with yet another image.

Step 1 – Define a shape. For this example a rectangle will be used.

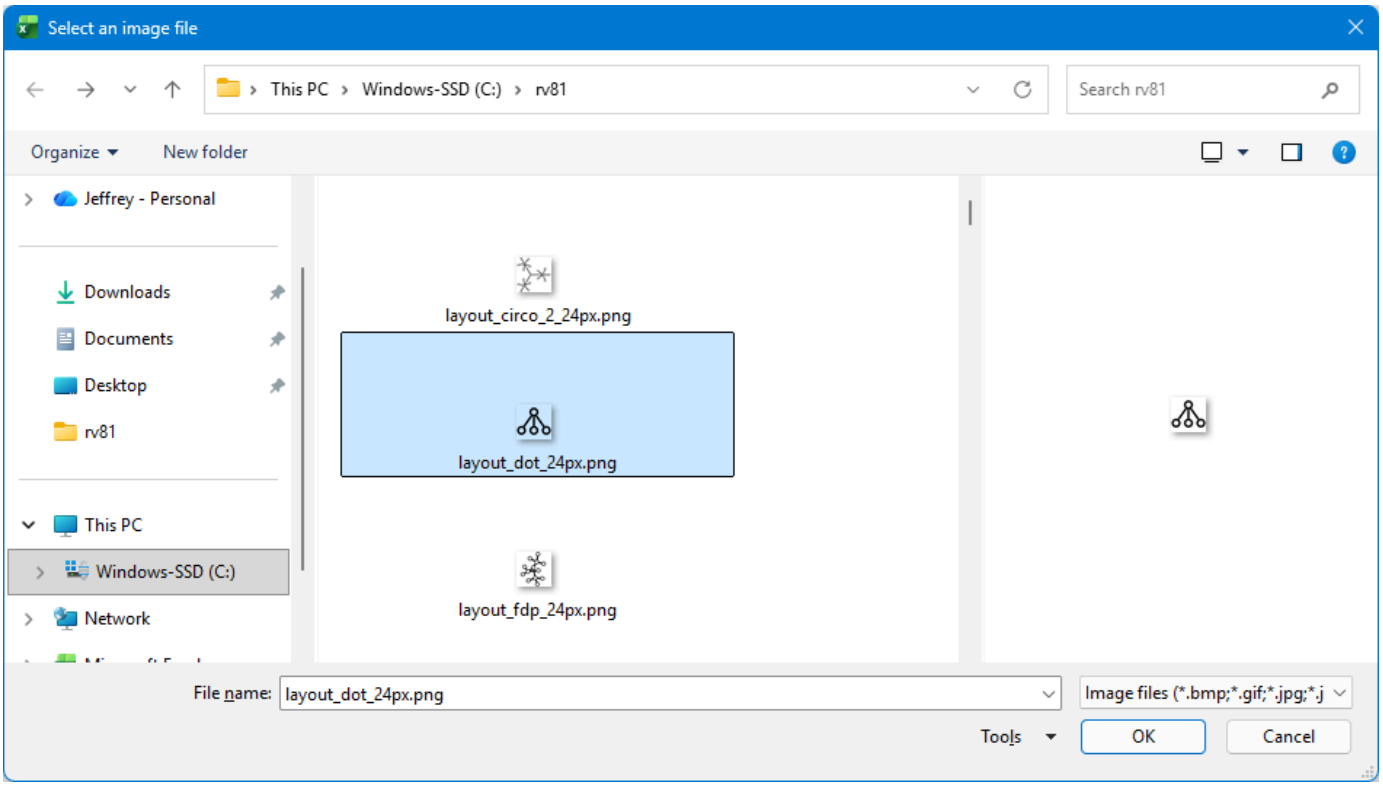


Step 2 – Look to the far right side of the Ribbon to find the image controls.



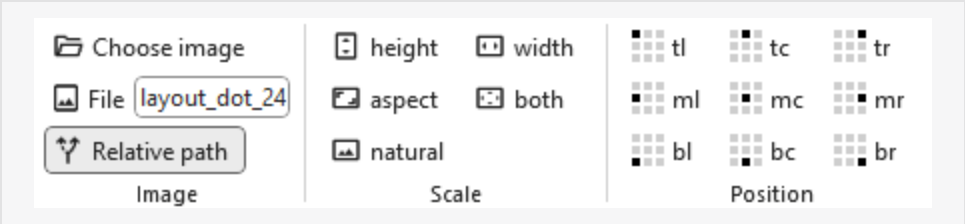
Press the `Choose Image` button.

Navigate to the directory containing the images and choose an image. A small image is selected in order to demonstrate scaling and placement.



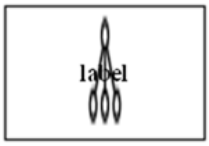
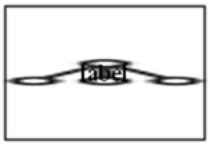
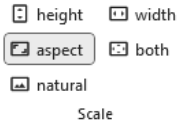
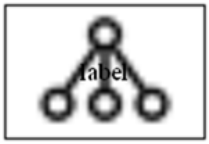
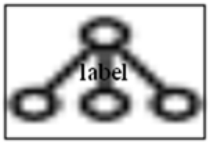
The image by default is placed in the center of the node. For example:

With the image selected, the Ribbon adapts to display additional options which can be used to scale the image, or position the image within the shape.



Scale the Image

Adjust the image by clicking the radio buttons in the **Scale** group. Only one button can be selected. If you make a second selection, your first selection is replaced.

Scale	Radio Button	Preview	Description
height			Stretch image to fill node height; width remains unchanged.
width			Stretch image to fill node width; height remains unchanged.
aspect			Uniformly scale image to fit node while preserving aspect ratio.
both			Stretch image to fill both width and height of node; aspect ratio may distort.
natural			Use image's natural size; node expands to fit (default).

No scaling (i.e., **natural** scaling) will be used in order to demonstrate how to position images which are smaller than the node.

Adjust the Image Position

The **Position** group contains nine toggle buttons that work in **radio button fashion**. Selecting any button automatically unselects the previously chosen option.

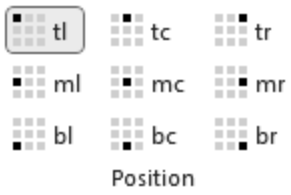
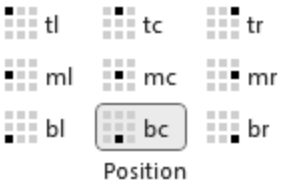
These buttons correspond to the nine possible locations where images can be placed relative to the cell or shape via the **imagepos** attribute:

+	Left	Center	Right
Top	`tl`	`tc`	`tr`
Middle	`ml`	`mc`	`mr`
Bottom	`bl`	`bc`	`br`

By default, when the **imagepos** attribute is omitted, the image is placed in the middle center.

You can reposition the image within the node by selecting a Position radio button, as shown below.

Position	Buttons pressed	Preview Image
Default		

Position	Buttons pressed	Preview Image
Top Left		
Bottom Center		

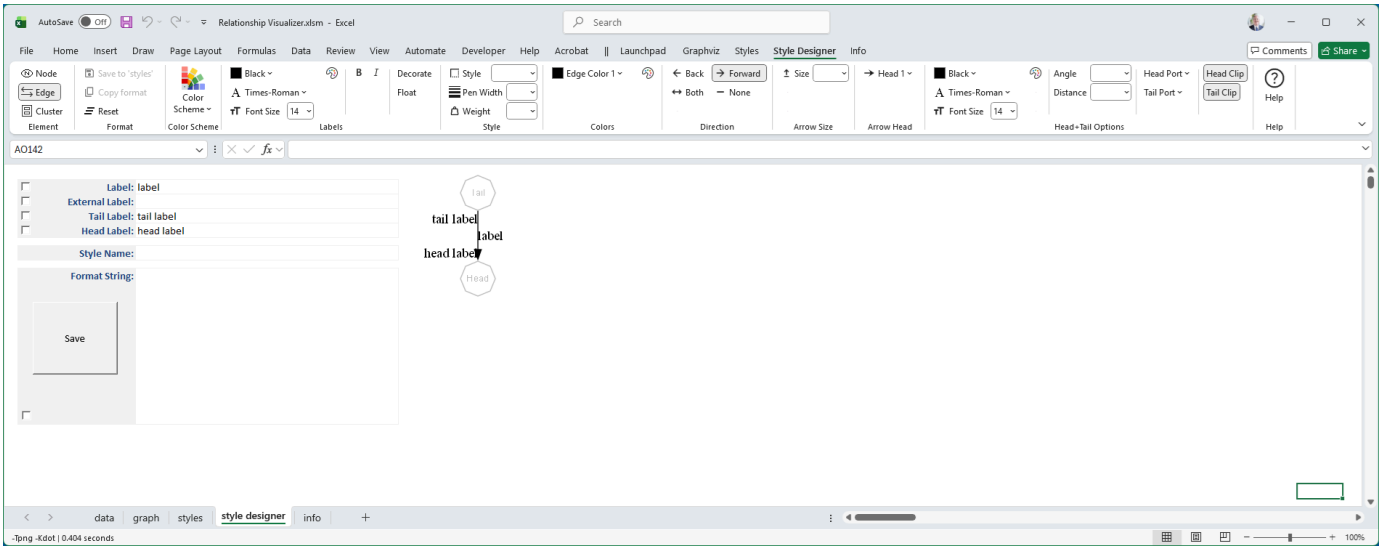
Edges

Edges can have styles just as nodes do.

To create an edge style definition in the **Style Designer** worksheet:

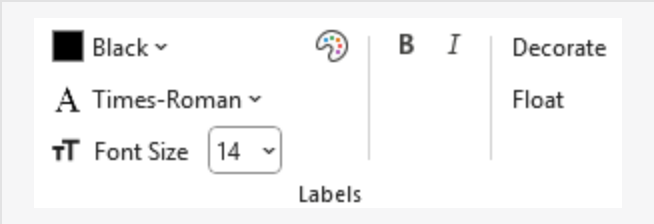
1. Change the **Design Mode Element** to **Edge**.
    - This switches the ribbon controls to attributes appropriate for edges (e.g., `color`, `style`, `penwidth`, `arrowhead`).
    - Node-specific attributes such as `shape` will no longer be available.
2. Press the **Reset** button.
    - This clears all style values carried over from node definitions.
    - Starting from a clean slate ensures that only edge-specific attributes are applied.
3. Use the ribbon, preview image, and format string just as you did for nodes.
    - The selected attributes are displayed in the ribbon.
    - The **Format String** is updated with edge attributes (e.g., `color=blue, style=dashed`).
    - The preview image shows exactly how Graphviz will render the edge.

The style designer worksheet appearance changes to look as follows:



Edge Labels

Labels for edges are specified in the same way as labels for nodes, with the same styling options (e.g., font, color, size).



However, edge labels include two additional toggle attributes:

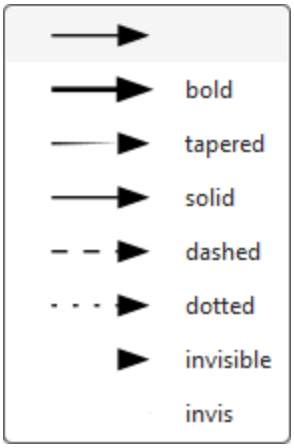
- **Decorate** - draws a line from the edge to its label, visually connecting the text to the edge.
- **Float** - allows the label to float freely near the edge rather than being anchored to a fixed position.

These options are available as toggle buttons in the **Style Designer** ribbon.
When selected, they are added to the **Format String** (e.g., `decorate=true, float=true`) and shown in the preview image.

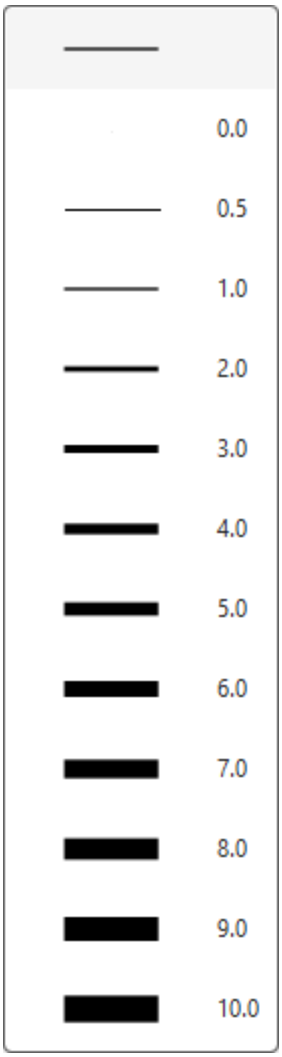
Edge Style

Edges can be styled using several attributes available in the **Style Designer** worksheet. These attributes control the visual appearance and relative importance of edges in the graph.

- **Style**
  - Defines the line pattern or effect applied to the edge.
  - Common values include `solid`, `dashed`, `dotted`, `bold`, and `tapered`.
  - For example, `style=dashed` produces a broken line, while `style=bold` thickens the edge for emphasis.



- **Penwidth**
  - Specifies the thickness of the edge line.
  - Larger values produce heavier lines, useful for highlighting important connections.
  - Example: `penwidth=2.0` doubles the default line thickness.



- **Weight**
  - Influences how strongly the edge affects the layout.
  - Higher weights encourage Graphviz to keep connected nodes closer together.
  - Example: `weight=5` makes the edge act like a stronger “spring” in the layout engine.

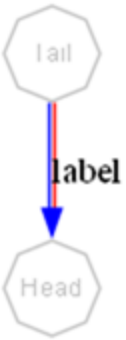
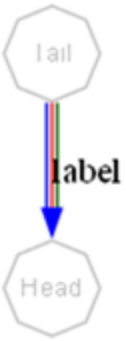
Edge Colors

Edge colors are specified in the same way as node colors, with support for both **color schemes** and the **RGB Color Dialog**.

- **One Color**
  - Apply one color to the edge line.
  - Example: `color=Blue`
- **Multiple Colors (up to 3)**
  - Specify up to three colors; Graphviz renders the edge as parallel lines in the given colors.
  - Example: `color="Blue:Red:DarkGreen"`
- **Color Schemes**
  - Choose colors from Graphviz’s predefined schemes (e.g., `rdbu11`, `greens3`).
  - Example: with `colorscheme=rdbu11`, use `color="2:4:6"` to reference indexed colors from that scheme.
- **RGB Color Dialog**
  - Select exact RGB values for precise customization beyond scheme defaults.

When you set edge colors, the chosen values are displayed in the **Style Designer** ribbon, added to the **Format String** (e.g., `color="red:yellow"`), and shown in the preview image.

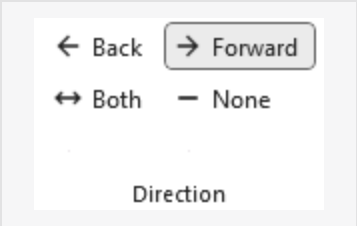
For example:

# Colors	Selection	Preview
1	Blue Edge Color 2 Colors	
2	Blue Red Edge Color 3 Colors	
3	Blue Red DarkGreen Colors	

Edge Direction

The Graphviz **dir** attribute controls the arrowheads drawn on an edge.

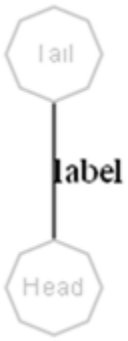
In the **Style Designer** worksheet, this is managed through four toggle buttons in the **Direction** group that act in radio button fashion. Selecting one option automatically clears the previous choice.



Supported values are:

- **forward**
  - Draws an arrowhead at the **target end** of the edge.
  - Provides Ribbon options for choosing **arrowhead styles** to be displayed.
  - Example: `dir=forward`
- **back**
  - Draws an arrowtail at the **source end** of the edge.
  - Provides Ribbon options for choosing **arrowtail styles** to be displayed.
  - Example: `dir=back`
- **both**
  - Draws arrowheads at **both ends** of the edge.
  - Provides Ribbon options for choosing both **arrowhead** (target end) and **arrowtail** (source end) styles to be displayed.
  - Example: `dir=both`
- **none**
  - Suppresses arrowheads entirely, leaving a plain line.
  - Removes the arrowhead and arrowtail style options from the ribbon.
  - Example: `dir=none`

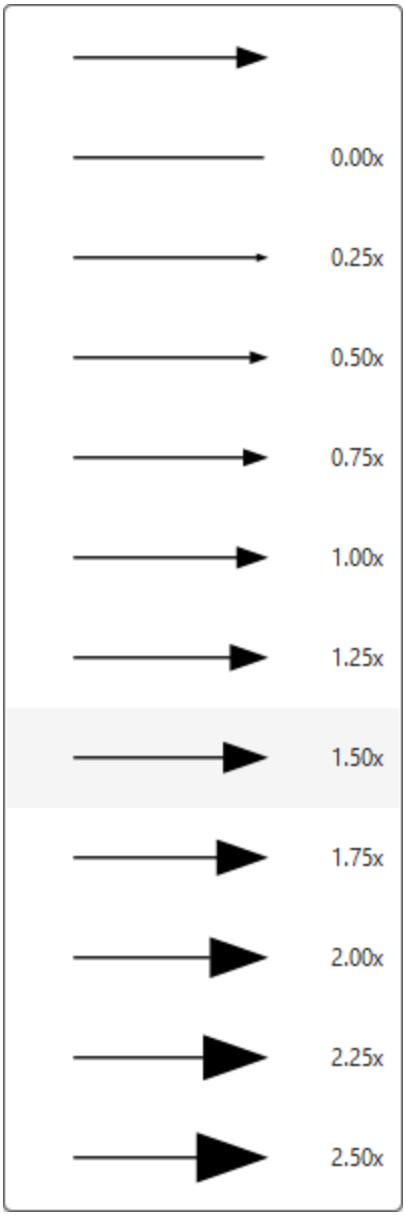
When you select a direction, the chosen value is displayed in the **Style Designer** ribbon, added to the **Format String** (e.g., `dir=both`), and shown in the preview image.

Direction	Ribbon	Preview
`dir=none`	← Back → Forward ↔ Both — None Direction	
`dir=forward`	← Back → Forward ↔ Both — None Direction | ↥ Size | → Head 1 Arrow Size | Arrow Head	
`dir=back`	← Back → Forward ↔ Both — None Direction | ↥ Size | ← Tail 1 Arrow Size | Arrow Tail	
`dir=both`	← Back → Forward ↔ Both — None Direction | ↥ Size | ← Tail 1 | → Head 1 Arrow Size | Arrow Tail | Arrow Head	

> Arrowhead and arrowtail styles will be described in detail in a later section.

Arrow Size

The **arrowsize** attribute scales the size of an edge's arrowhead or arrowtail.



It acts as a simple multiplier applied to the base size of the selected arrow style.

- **Value**
  - A numeric scaling factor (default is `1.0`).
  - Values greater than 1 enlarge the arrow; values less than 1 reduce it.
  - Example: `arrowsize=1.5` makes the arrowhead 50% larger than normal.

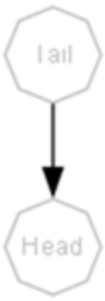
- **Effect on Direction**
  - When `dir=forward`, the scaling applies to the **arrowhead**.
  - When `dir=back`, the scaling applies to the **arrowtail**.
  - When `dir=both`, the scaling applies to **both ends**.
  - When `dir=none`, the attribute has no visible effect.

The selected value is shown in the **Style Designer** ribbon, added to the **Format String** (e.g., `arrowsize=0.75`), and reflected in the preview image.

Arrow Heads

Arrowheads are a popular styling option for edges, and Graphviz provides a robust set of shapes to choose from.

You may stack multiple arrowhead types to create custom designs. For example:

1 Arrow Head	2 Arrow Heads	3 Arrow Heads
		
`arrowhead="normal"`	`arrowhead="normalodot"`	`arrowhead="normalodotcurve"`

The Relationship Visualizer ribbon supports up to **three stacked arrowheads**:

- When you choose the **first** arrowhead, a **second dropdown** automatically appears.
- After selecting a **second** arrowhead, a **third dropdown** becomes available.
- Each dropdown represents one position in the stack, allowing you to build compound arrowhead shapes.

These selections apply to the end(s) of the edge based on the `dir` setting (e.g., `forward`, `back`, or `both`).

Arrowhead and arrowtail styles are added to the **Format String** (e.g., `arrowhead=diamond`, `arrowtail="dotvee"`), and the preview updates accordingly.

Each change updates the **Edge Format String** and renders a sample graph showing how the edge will appear based on the current **Layout Engine** and **Splines** settings on the `settings` worksheet.

Be aware that the visual result may vary depending on how different layout engines handle splines, head ports, and tail ports.

Arrow Tails

Arrowtails behave the same way as arrowheads: you may stack up to three tail shapes to create compound designs, and each selection reveals the next dropdown in the sequence. The stacked arrowtails apply to the source end of the edge whenever the direction setting (`dir=back` or `dir=both`) enables them.

With both arrowheads and arrowtails available, you can combine these glyphs in a wide variety of ways. The following gallery shows the full set of arrow shapes that Graphviz supports, which you can mix, match, and stack to create custom edge endpoints.

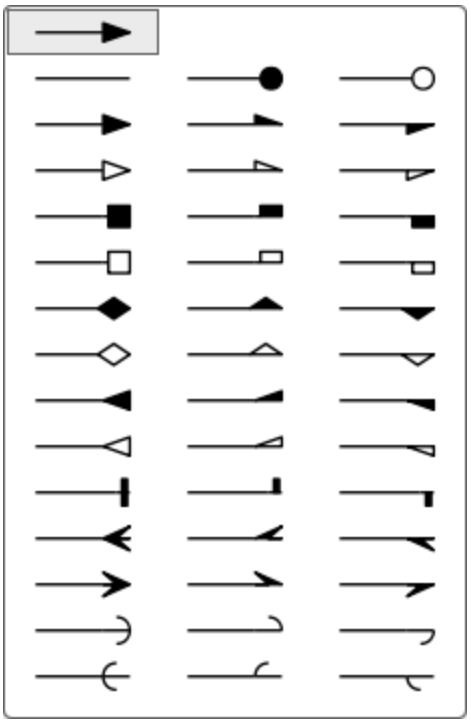
Arrow tails have the same set of choices as arrow heads. Like arrow heads, Relationship Visualizer allows you to choose up to 3 styles for Arrow Tails.

Arrow Glyphs

Graphviz provides a rich collection of arrowhead and arrowtail glyphs that you can use individually or stack to create custom edge endpoints. Each glyph has its own visual character, ranging from simple geometric shapes to more expressive markers. Stacking

them allows you to build complex designs that convey direction, emphasis, or semantic meaning.

The following gallery shows the complete set of arrow glyphs supported by Graphviz. These shapes can be used for both **arrowheads** and **arrowtails**, and up to three may be combined in sequence to form a compound style. Use this reference to explore the available options and choose the combinations that best fit your diagram's purpose.



The arrowhead and arrowtail glyphs shown above use the standard Graphviz names. From left to right, each glyph is labeled with its corresponding attribute value:

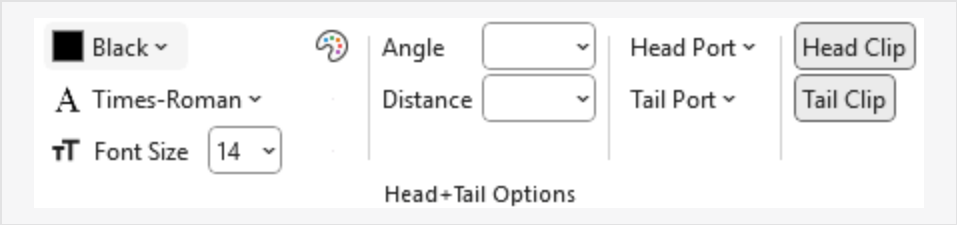
Left + Right	Left Only	Right Only
`none`	`dot`	`odot`
`normal`	`lnormal`	`rnormal`
`onormal`	`olnormal`	`ornormal`
`box`	`lbox`	`rbox`
`obox`	`olbox`	`orbox`

Left + Right	Left Only	Right Only
`diamond`	`ldiamond`	`rdiamond`
`odiamond`	`oldiamond`	`ordiamond`
`inv`	`linv`	`rinv`
`oinv`	`olinv`	`orinv`
`tee`	`ltee`	`rtee`
`crow`	`lcrow`	`rcrow`
`vee`	`lvee`	`rvee`
`curve`	`lcurve`	`rcurve`
`icurve`	`licurve`	`ricurve`

Use these names when specifying `arrowhead`, `arrowtail`, or stacked combinations such as `arrowhead="dotvee"` or `arrowtail="diamondtee"`.

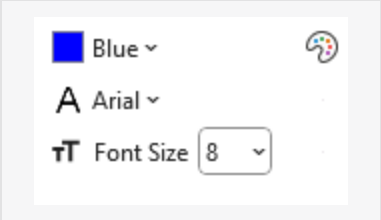
Head & Tail Options

These controls provide assistance in defining the head and tail attributes for an edge.

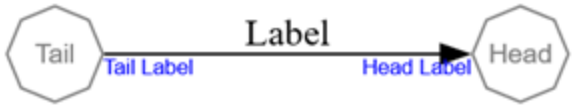


Label Font Color - Label Font Name - Label Font Size

These attributes provide a way to differentiate the text at the end of the edges where they meet the node.



Appears as:



With Format string:

```
labelfontname="Arial" labelfontsize="8" labelfontcolor="Blue"
```

Label Angle

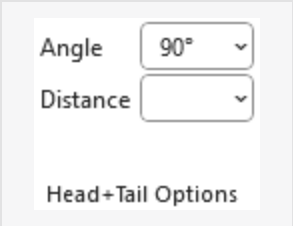
`labelangle=` controls the **direction** in which a head label or tail label appears around the point where the edge touches the node.

Imagine standing at the spot where the edge meets the node. Now imagine a line pointing straight back along the edge. That's the starting direction (0 degrees).
`labelangle` tells Graphviz how far to rotate from that starting direction:

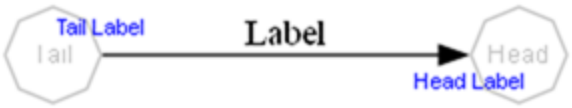
- **Positive numbers** rotate the label **to the left**
- **Negative numbers** rotate the label **to the right**

By changing the angle, you choose which "side" of the node the label appears on.

For example, setting the label angle to 90 degrees:



Appears as:



With Format String:

```
labelangle=90 labelfontname=Arial labelfontsize=8 labelfontcolor=Blue
```

Label Distance

`labeldistance=` controls **how far away** the head label or tail label appears from the point where the edge touches the node.

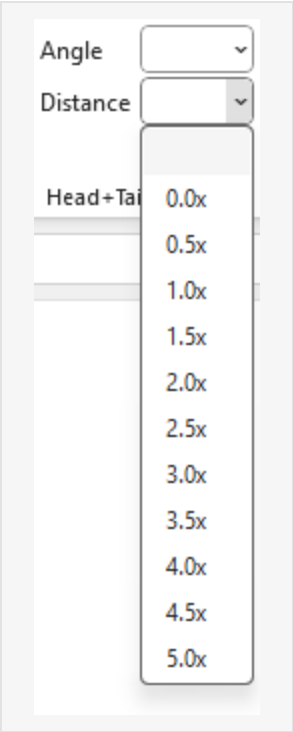
Instead of setting the distance directly in points, `labeldistance` acts as a **scaling factor**. Graphviz starts from a built-in base distance (about **10 points**), and your value multiplies

that distance:

- A value of **1.0** keeps the default spacing
- A value of **2.0** places the label about twice as far away
- A value of **0.5** moves it to about half the default distance

A **point** is a standard typographic unit: there are **72 points in one inch**, so these changes adjust the label's distance in small, predictable steps.

In short, `labeldistance` tells Graphviz to move the label **closer or farther** from the node by scaling the default distance.



For example, `labeldistance=3` appears as:

With Format String:

```
labelangle=90 labeldistance=3 labelfontname=Arial labelfontsize=8 labelfontcolor=Blue
```

Label Angle & Label Distance

Used together, `labelangle` and `labeldistance` let you control both **where** a label appears around the node and **how far out** it sits. `labelangle` chooses the direction (left, right, above, below, or anywhere in between) while `labeldistance` scales the default spacing to move the label closer or farther away. Adjusting both gives you precise, intuitive control over label placement at the point where the edge meets the node.

This example depicts when `labelangle=` and `labeldistance=` attributes are used together.

Appears as:

With Format String:

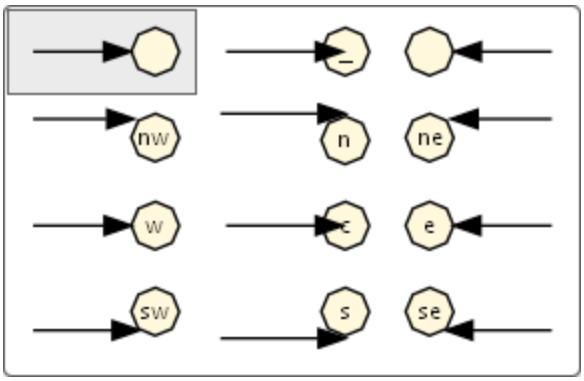
```
labelangle=90 labeldistance=3 labelfontname=Arial labelfontsize=8 labelfontcolor=Blue
```

Head Port

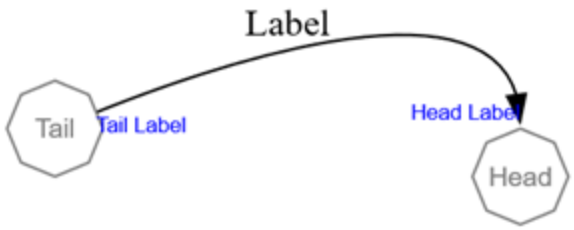
Indicates where on the head node to attach the head of the edge. In the default case, the edge is aimed towards the center of the node, and then clipped at the node boundary.

If a compass point is used, it must be one of the following: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, `c`, or `_`. A compass point adjusts the edge's attachment point so that it aims for the

specified location on the port, or, if no port name is provided, on the node itself. The compass point `c` targets the center of the node or port. The compass point `_` instructs Graphviz to choose the side of the port that lies on the exterior of the node; if no such side exists, the center is used instead. When a port name is supplied without a compass point, the default value is `_`.



Appears As:



With Format String:

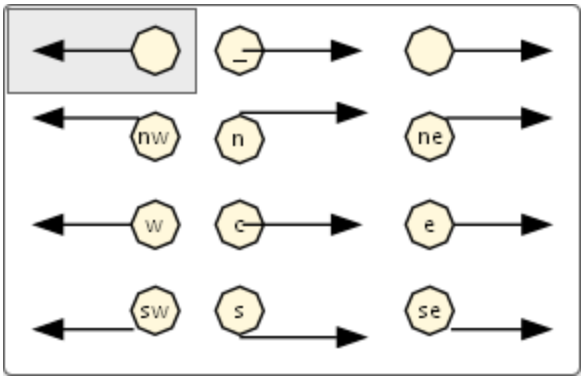
```
labelfontname=Arial labelfontsize=8 labelfontcolor=Blue headport=n
```

Tail Port

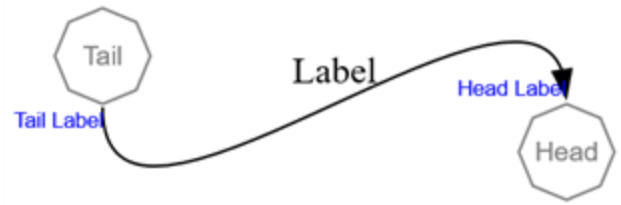
Indicates where on the tail node to attach the tail of the edge.

If a compass point is used, it must be one of the following: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, `c`, or `_`. A compass point modifies edge placement so that the edge aims for the specified point on the port, or, if no port name is supplied, on the node itself. The compass point `c` targets the center of the node or port. The compass point `_` indicates that Graphviz should choose the side of the port that lies on the exterior of the node; if no such

side exists, the center is used instead. When a port name is provided without a compass point, the default compass point is `_`.



Appears as:



With Format String:

```
labelfontname=Arial labelfontsize=8 labelfontcolor=Blue headport=n tailport=s
```

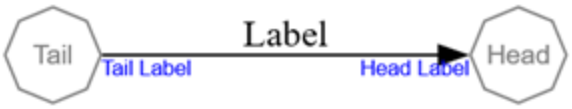
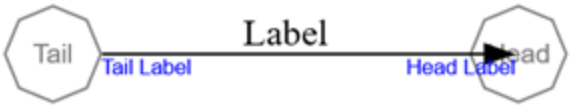
Clipping Behavior

Graphviz uses *clipping* to decide how far an edge (its spline and arrowhead) runs into a node. The attributes `headclip` and `tailclip` control this behavior independently for the head and tail ends of an edge. These settings affect the **edge and arrowhead**, not the label text itself.

Head Clip

`headclip=` controls how the edge is clipped at the **head** node.

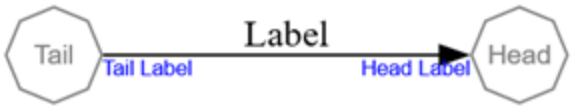
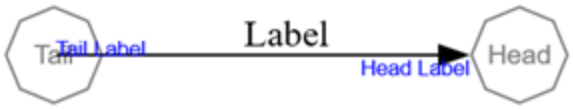
- When `headclip=true` (the default), Graphviz clips the spline at the boundary of the head node. The arrowhead sits at the edge of the node shape, rather than running into the center.
- When `headclip=false`, the edge is not clipped to the node boundary. The spline and arrowhead may extend into the interior of the node, often aiming at its center.

Buttons	Preview
Head Clip Tail Clip	
	Edge head **is** clipped at the node
Head Clip Tail Clip	
	Edge head **is not** clipped at the node

Tail Clip

`tailclip=` controls how the edge is clipped at the **tail** node.

- When `tailclip=true` (the default), Graphviz clips the spline at the boundary of the tail node. The edge meets the node at its outline.
- When `tailclip=false`, the spline is allowed to extend into the node, so the edge may appear to start from a point inside the node.

Buttons	Preview
Head Clip Tail Clip	
	Edge tail **is** clipped at the node
Head Clip Tail Clip	
	Edge tail **is not** clipped at the node

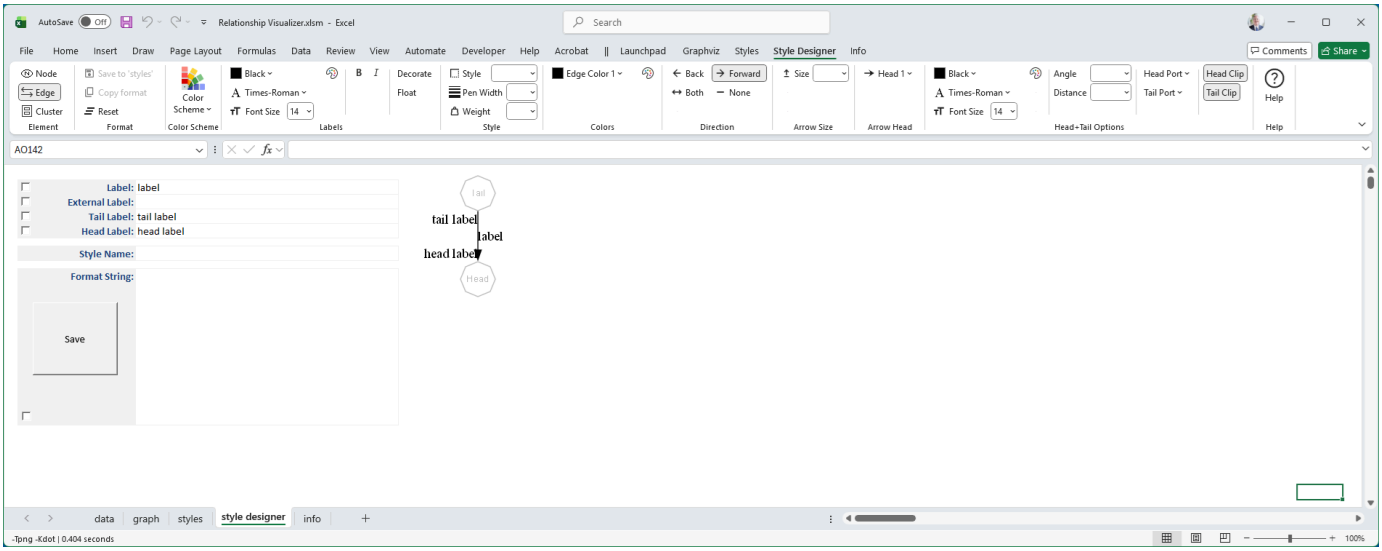
Clusters

Clusters can have styles just as nodes and edges do.

To create a cluster style definition in the **Style Designer** worksheet:

1. Change the **Design Mode Element** to **Cluster**.
   - This switches the ribbon controls to attributes appropriate for clusters (e.g., `Labels`, `Borders`, `Fill`).
   - Node-specific attributes such as `shape` will no longer be available.
   - Edge-specific attributes such as `arrowhead`, `arrowtail` will no longer be available.
2. Press the **Reset** button.
   - This clears all style values carried over from node definitions.
   - Starting from a clean slate ensures that only cluster-specific attributes are applied.
3. Use the ribbon, preview image, and format string just as you did for nodes and edges.
   - The selected attributes are displayed in the ribbon.
   - The **Format String** is updated with cluster attributes (e.g., `color=blue, style=dashed`).
   - The preview image shows exactly how Graphviz will render the cluster. There are seven shapes of various sizes to simulate how a cluster might be used.

The Style Designer worksheet appearance changes to look as follows:



Color Scheme

The cluster can specify a color scheme. Note that since clusters are subgraphs, you should use care in specifying a color scheme as nodes can inherit this specification.

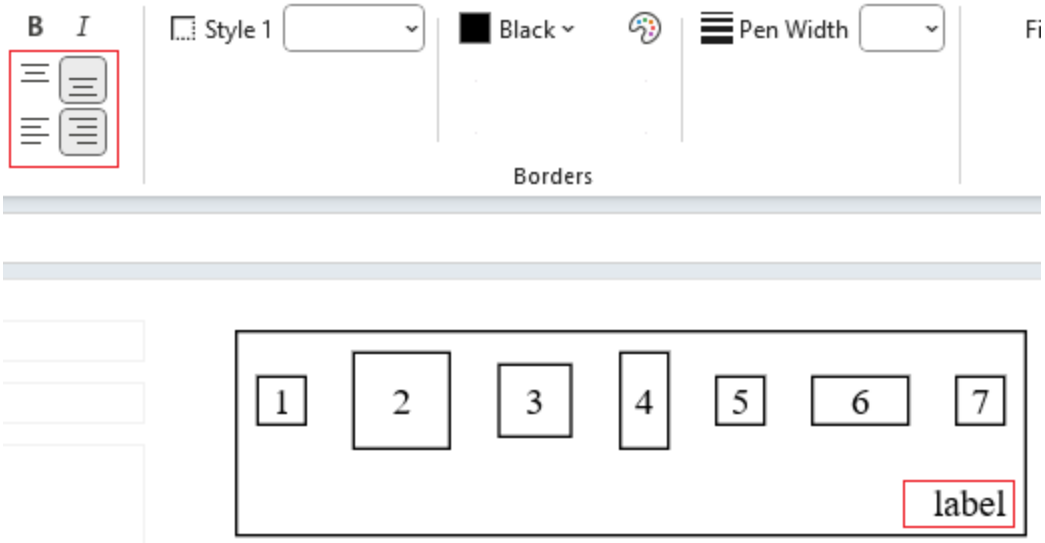
Labels

Font color, name, and size are specified for the cluster label the same way they are specified for nodes and edges. See Labels for more information.

Label Alignment

Nodes allow you to left-justify, center, or right-justify their labels. Clusters extend this by also allowing **top** or **bottom** alignment. These options are additive, so you can combine them to specify positions such as **bottom right**, **top left**, or **top center**.

The following example shows the alignment buttons set to place the cluster label in the bottom right of the cluster:



Borders

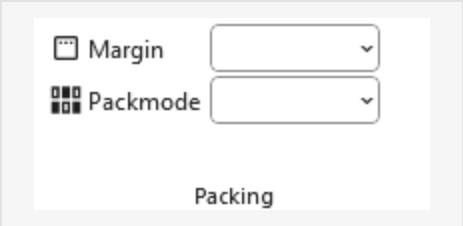
Border styles, colors, and pen widths for a cluster's rectangle are defined the same way they are for nodes. This means you can use the same attributes such as `color`, `penwidth`, and `style` to control how the cluster's outline looks. The cluster simply draws a rectangular boundary around its contents, and these attributes determine how that boundary is rendered. For more details on available options, see the Borders topic.

Fill Colors

Fill colors and gradient fills for a cluster's rectangle are defined the same way they are for nodes. You can use the same attributes such as `fillcolor`, `color`, and `style=filled`, or gradient specifications, to control how the cluster's background is rendered. The cluster simply applies these settings to its rectangular boundary. For details on available options, see the Fills topic.

Packing Options

If the layout on the Graphviz ribbon tab is set to the osage layout, an additional **Packing** group of controls will appear, as shown in the example below:

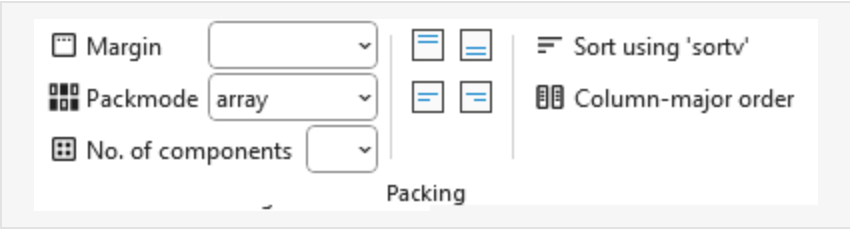


These controls let you adjust how nodes within clusters, or clusters within clusters, are arranged relative to one another in the final layout.

Two options are provided:

- **Margin** - Using the `pack` attribute, sets the amount of empty space between packed clusters. Values from 1/8" to 1" are available, in increments of 1/8". Larger margins spread clusters farther apart; smaller margins bring them closer together. Distances are converted from inches to points automatically.
- **Packmode** - Determines the strategy Graphviz uses when arranging clusters. Different modes influence whether clusters are packed tightly, aligned in rows or columns, or arranged using more geometric rules. The available modes are:
  - **clust** - Packs clusters based on their natural cluster structure, keeping related groups visually close.
  - **array** - Packs clusters into a grid-like arrangement.

  When **array** is selected, the Packing group expands to provide additional controls, as shown below:



Additional choices include:

- **No. of components** - How many components (nodes or clusters) to place before starting a new row or column.
- **Node alignment** - How nodes should be aligned within each cluster:
  - Top → `packmode=array_t`
  - Middle → `packmode=array`
  - Bottom → `packmode=array_b`
  - Left → `packmode=array_l`
  - Right → `packmode=array_r`
- **Sort using 'sortv'** → `packmode=array_u` - Sorts components based on their `sortv` attribute.
- **Column-major order** → `packmode=array_c` - Lays out components column-by-column instead of row-by-row.

`packmode` array flags can be combined to apply multiple effects at once. For example, the selections shown below:

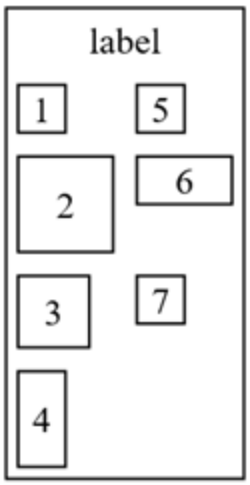
produce the Format String `pack=9 packmode=array_ctlu4` which breaks down as follows:

- `pack=9` - Sets the margin around the nodes to 1/8".
- `packmode=array_ctlu4` - Applies several array-mode flags at once:
  - `u` - Sort components by their `sortv` attribute.
  - `c` - Use column-major order.
  - `l` - Align nodes along the **left** edge within each column.
  - `t` - Align nodes along the **top** edge within each row.
  - `4` - Each column can contain **4** node or cluster objects (this example shows nodes).

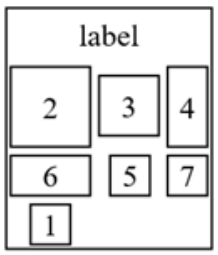
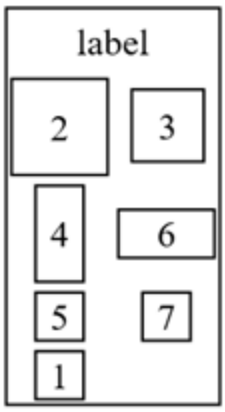
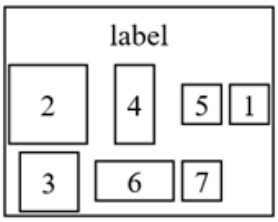
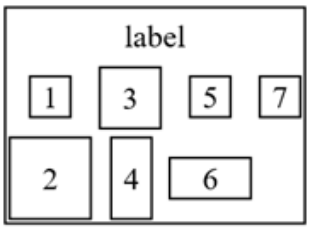
Together, these flags sort the nodes, arrange them column-by-column with four rows per column, and align them to the top and left within the grid.

The preview of the cluster appears as:

`pack=9 packmode=array_ctlu4`



Here are additional examples:

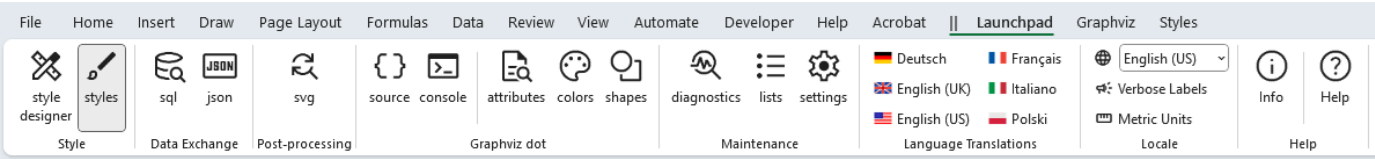
`packmode=clust`	`packmode=array_2`	`packmode=array_c2`	`packmode=array_cu2`
			
Best fit distribution	2 Columns in row major order	2 Rows in column major order	2 Rows in column major order, sorted

Style Gallery

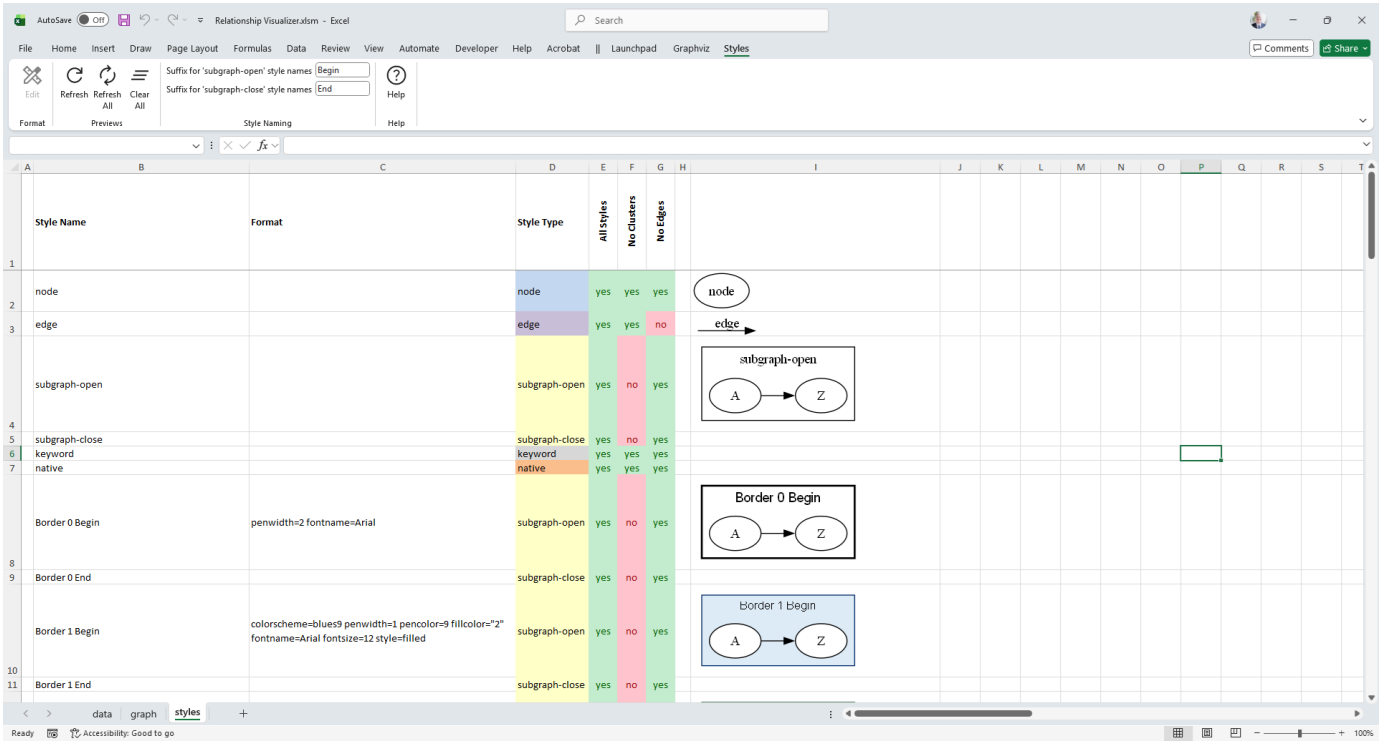
A key component of the Relationship Visualizer is the `styles` worksheet, where you can create style definitions for nodes, edges, and clusters. Conceptually, it works much like an HTML Cascading Style Sheet: you define a style name and specify how that style should appear (shape, color, font, and so on). Once defined, a style can be applied to any number of nodes or edges in the `data` worksheet.

The `styles` Worksheet

The `styles` worksheet is accessed from the **Style** section of the Launchpad ribbon tab.



The default `styles` worksheet appears as follows:



The `data` Worksheet has the following columns:

A	B	C	D	E (and beyond)	Last switch column + 2
Indicator	Style	Format	Style Type	View Switches	Preview Image

The columns are as follows:

(A) Indicator

Allows you to place a `#` character to denote a comment. This can be used to comment out a style so it is excluded from the renderings.

(B) Style

Specifies the style name.

(C) Format

Contains the style definition pasted from the `style designer` worksheet. This definition determines the visual appearance of any graph elements associated with this style in the `data` worksheet.

(D) Style Type

Must contain one of the following values: `node`, `edge`, `subgraph-open`, `subgraph-close`, `keyword`, or `native`. This value tells the Relationship Visualizer macros how to interpret the row and convert it into the appropriate DOT language commands.

(E) View Switches

Used for creating different views of the data. Each column must contain `Yes` or `No` to indicate whether the style should be included in the graph. These columns are described further in Creating Views.

All spreadsheets created from the Relationship Visualizer Excel template include a default Column E labeled **All Styles**, with all style switches set to `Yes`. When this column controls the view, every style is included in the graph.

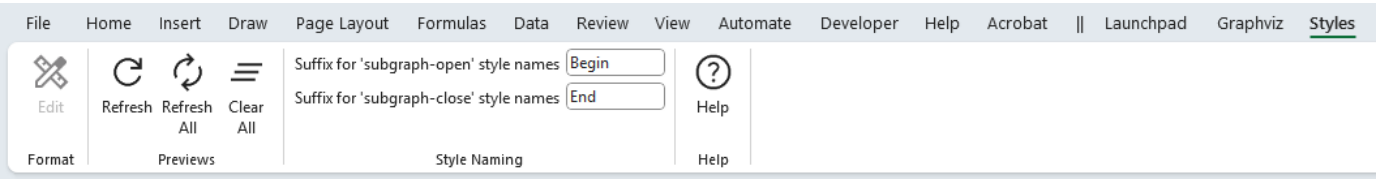
(I) Preview Image

A preview image of the style can be placed after the last view column. These preview images are generated using the Styles ribbon tab.

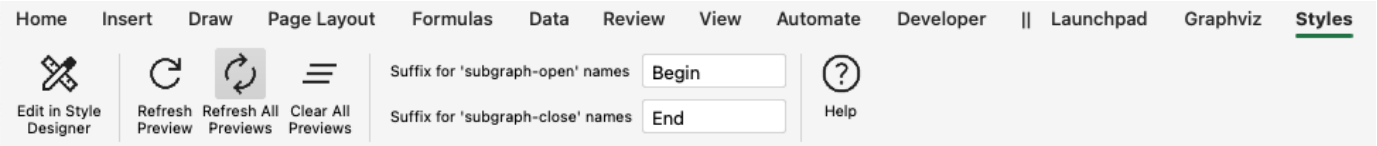
The `Styles` Ribbon Tab

The `Styles` ribbon tab is activated whenever the `styles` worksheet is activated. It appears as follows:

Windows

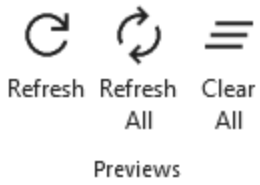
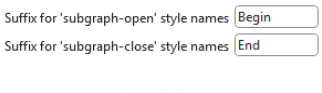
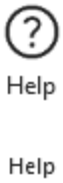


macOS



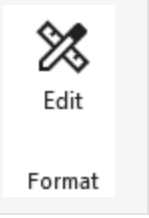
It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
Format	Edit Format	Facilitates editing the style format.

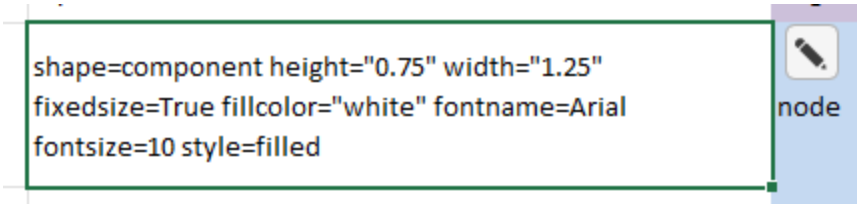
Group	Controls	Description
Previews		Provides the action buttons for managing the style preview images on the `Styles` worksheet.
Style Naming		Allows you to specify the suffix for the end of the style name to denote when the cluster begins and ends.
Help		Provides a link to the `Help` content for the `styles` worksheet (i.e. this web page).

Format

The `Format` group controls facilitate editing the style format.



Label	Control Type	Description
Edit	 Button	The **Edit** button becomes enabled whenever a single format cell in Column C is selected, and the row defines a style for a node, edge or cluster. When pressed, the format string in that cell is parsed, and the resulting values are used to initialize the Style Designer worksheet, where the style definition can be edited. A second location where the **Edit** button appears is as a floating button on the right side of any selected format cell. This button displays a pencil

Label	Control Type	Description
		icon, as shown in the following example:  Clicking the pencil button performs the same action as selecting the **Edit** button in the Ribbon.

Previews

The **Previews** group provides the action buttons used to manage the style preview images on the `Styles` worksheet.

Label	Control Type	Description
Refresh	Button	Creates a new preview image for the current row. This is useful when you manually modify a single style definition and want to see the updated appearance.
Refresh All	Button	Deletes all preview images on the `styles` worksheet and generates a completely new set. This is useful when you make a bulk change to *all* style definitions, such as updating a font name or size.
Clear All	Button	Deletes all the images on the `styles` worksheet.

Style Naming

Two rows are created when you use the `style designer` to define a style for a cluster. These settings allow you to specify the suffix appended to the style name to indicate where the cluster begins and ends.

The default suffixes are **" Begin"** and **" End"**, but you may choose alternatives such as **" Start"/" Stop"** or **" Open"/" Close"**.

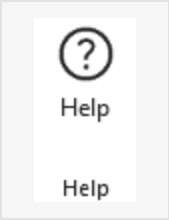
These values are also used by the `sql` worksheet when emitting rows when `CLUSTER` and `SUBCLUSTER` clauses are used in the `SQL` statement. See the sql topic for more information.

Two rows are created when you use the `style designer` to define a style for a cluster. These settings specify the suffix appended to the style name to indicate where the cluster begins and ends. The default suffixes are **"Begin"** and **"End"**, but you may choose alternatives such as **"Start"/"Stop"** or **"Open"/"Close"**.

These suffix values are also used by the `sql` worksheet when generating rows for `CLUSTER` and `SUBCLUSTER` clauses in an `SQL` statement. For more details on how SQL-driven clustering works, see the Grouping Data into Clusters and Subclusters topic.

Label	Control Type	Description
Suffix for 'subgraph-open' style names	Text Edit	Suffix to append to cluster names to indicated the beginning of a cluster.
Suffix for 'subgraph-open' style names	Text Edit	Suffix to append to cluster names to indicated the end of a cluster.

Help



Provides a link to the `Help` content for the `Info` worksheet (i.e. this web page).

Label	Control Type	Description
Help	Button	Provides a link to this web page.

Create Views

The Relationship Visualizer is capable of handling much larger data sets than can be easily viewed. As data sets grow, it often becomes useful to focus on specific portions of the information. The Relationship Visualizer supports this by providing view controls on the `styles` worksheet, where **Yes** and **No** switches determine which data styles are included in the graph.

Views vs. Layers

In Relationship Visualizer terminology, the term **Views** refers to redrawing the graph using only a subset of nodes and edges. Because Graphviz optimizes layout based on the visible connections, the positions of shapes often change when a view is applied.

This differs from **Layers**, a concept used in tools such as Microsoft Visio, where shapes can be assigned to layers that are simply shown or hidden. In a layered system, shapes do not move; they are merely visible or invisible. Graphviz also supports a concept of layers, but Graphviz layers are outside the scope of this topic.

The Relationship Visualizer's `styles` worksheet defines three columns for each style definition.

1. The **All Styles** column contains **Yes** for every style definition.
2. The **No Clusters** column contains **No** for any style whose Style Type is `subgraph-open` or `subgraph-close`.
3. The **No Edges** column contains **No** for any style whose Style Type is `edge`.

When the graph is created, any row whose Style value is **Yes** is included in the graph, and any row whose Style value is **No** is omitted.

The column used to make these decisions is selected on the Graphviz tab in the `Graph View` dropdown list. The **All Styles** column is the default view. With **All Styles** selected, the

graph appears as shown below.

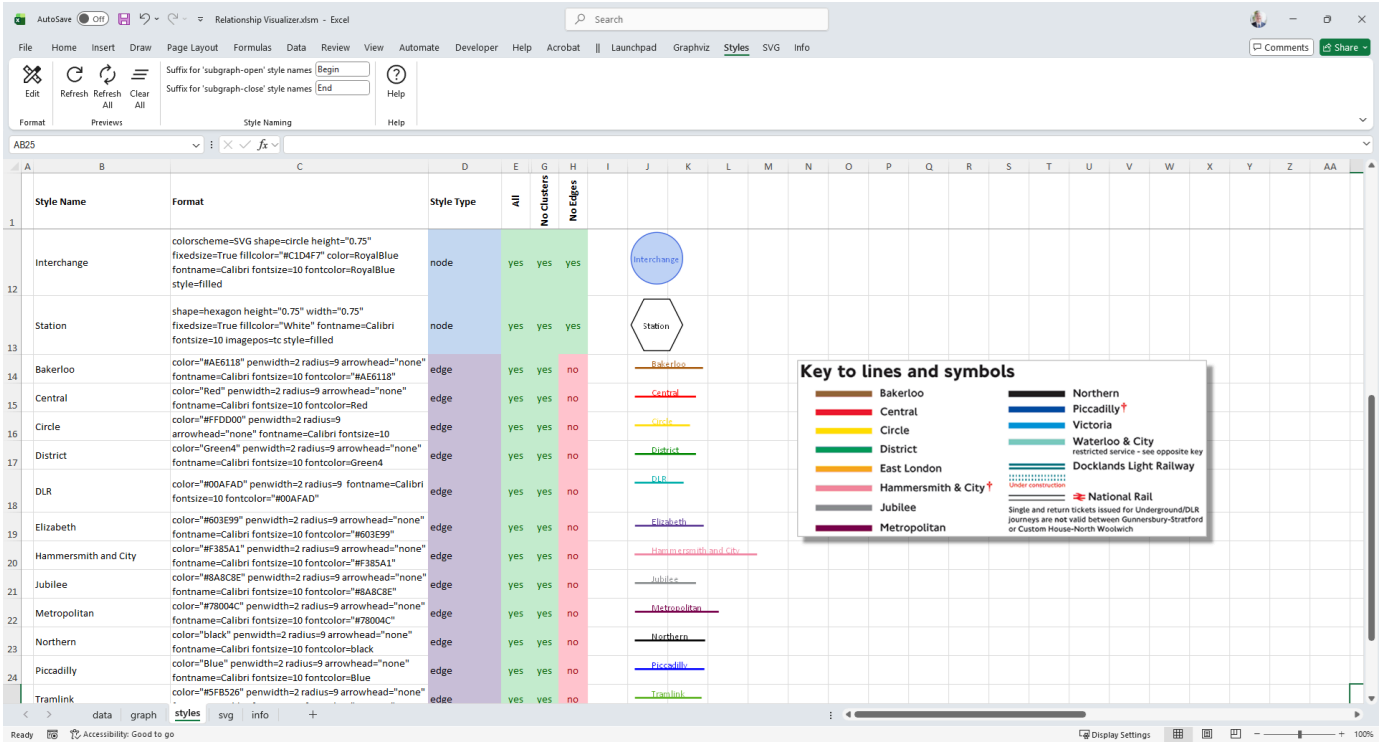
If `Graph View` is changed to **No Clusters** the graph is redrawn and appears as

---

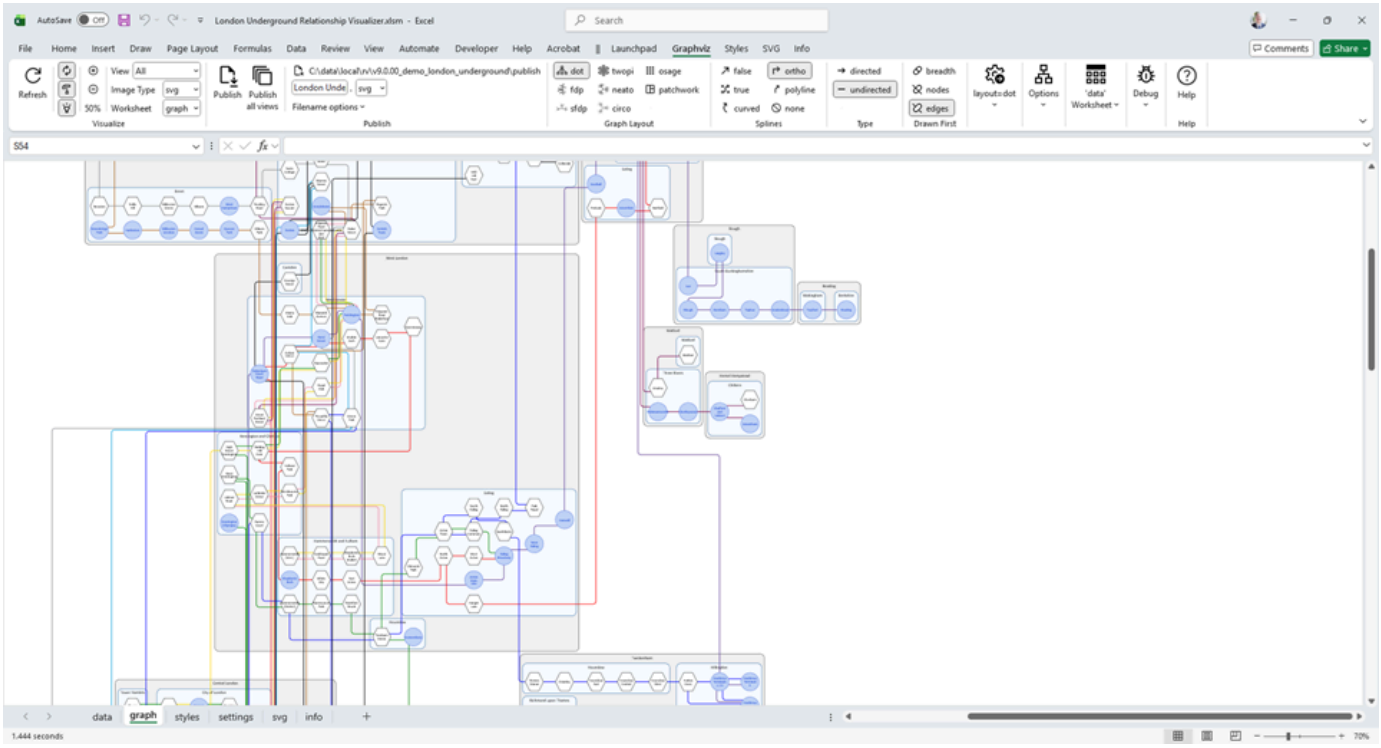
Custom View Example

The Overview page shows several illustrations of a graph of the London Underground.

We began by creating edge styles using the official London Underground line colors.



These styles were applied to a data set of the subway stations and the station-to-next-station relationships that connect each station to the next. The graph of the complete London Underground appears as:



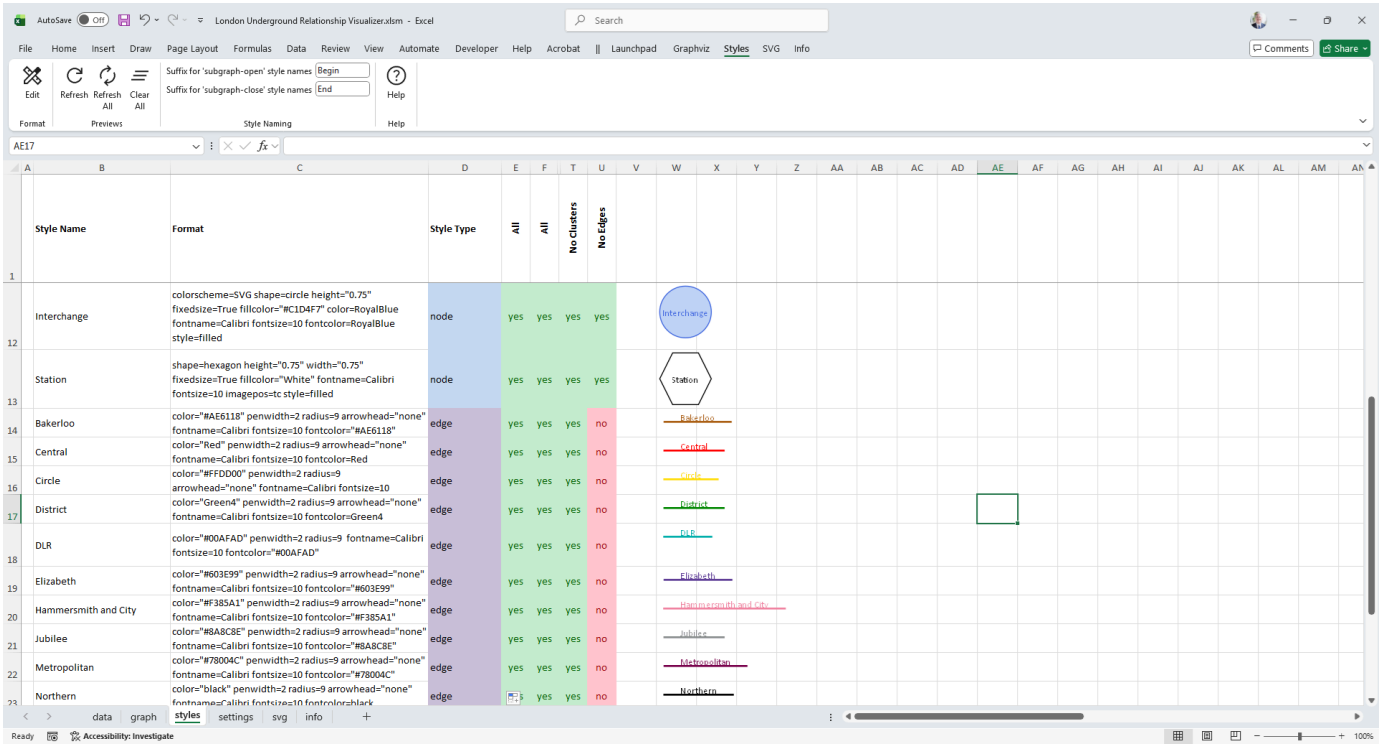
This gives us a full data set and a complete graph, with each connection styled according to its line color. But what if we want to view only a single subway line?

To do this, we define a column of View switches in the `styles` worksheet and instruct the Relationship Visualizer to generate the graph using only the styles enabled in that column.

Let us create a view that restricts the graph to only the **Bakerloo** subway line relationships.

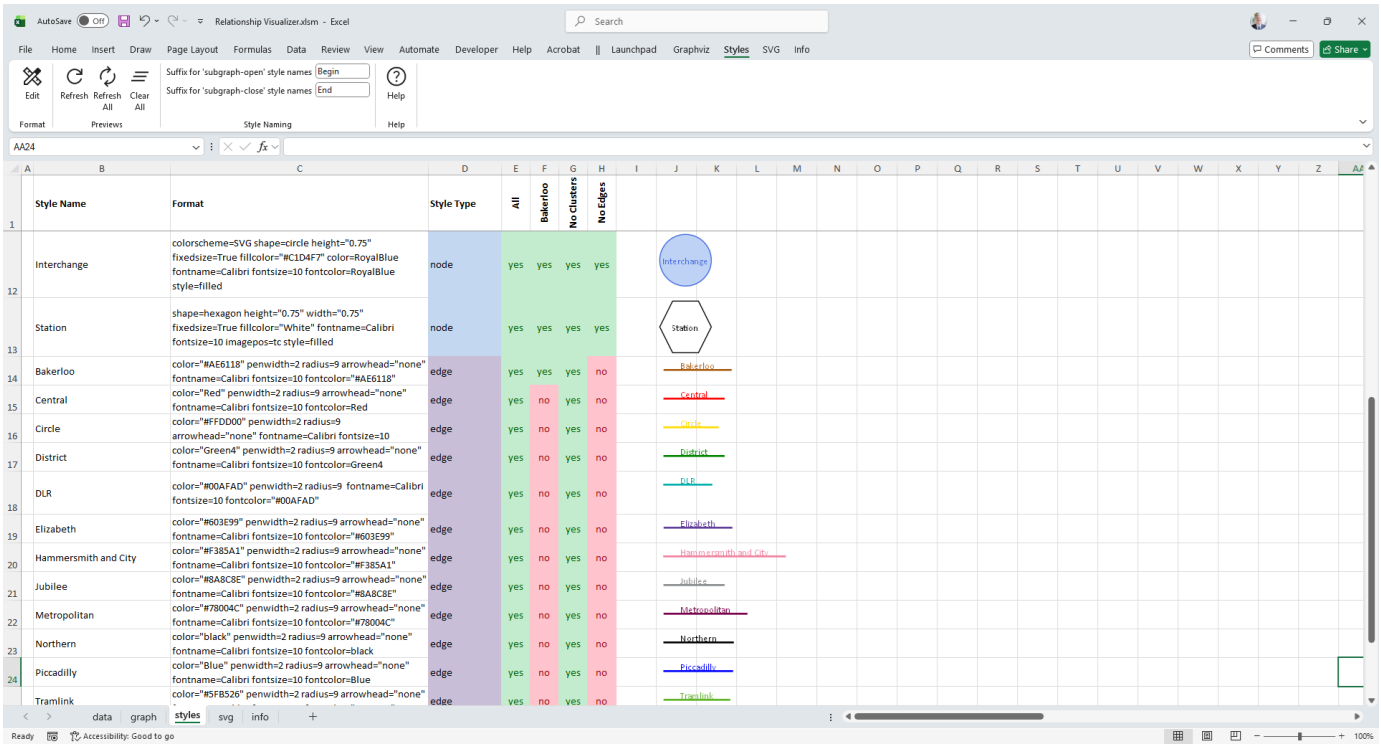
Step 1 — Switch to the `styles` worksheet.

Step 2 — Copy Column E (with the heading **All**) and insert the copy as Column F. The `styles` worksheet will then appear as follows:

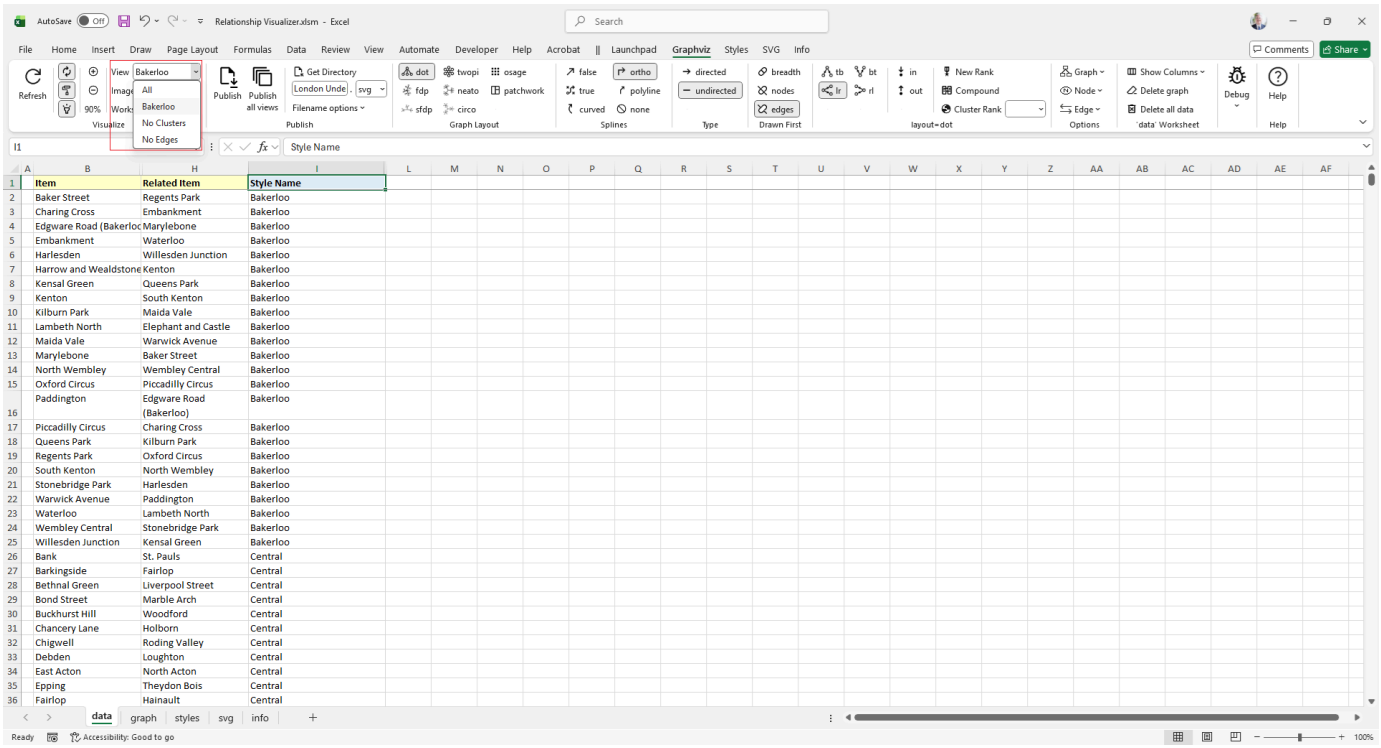


Step 3 — Change the heading in cell F1 to **Bakerloo**, and change all the switches in Column F from **Yes** to **No** for every row where the `Style Type` is `edge`, **except** for the row whose style name is **Bakerloo**.

The `styles` worksheet should now appear as:



Step 4 — We now need to change a setting on the `Graphviz` ribbon tab so the Relationship Visualizer uses only the styles enabled with **Yes** in Column F. Switch to the `data` worksheet. The `data` worksheet should appear as follows:

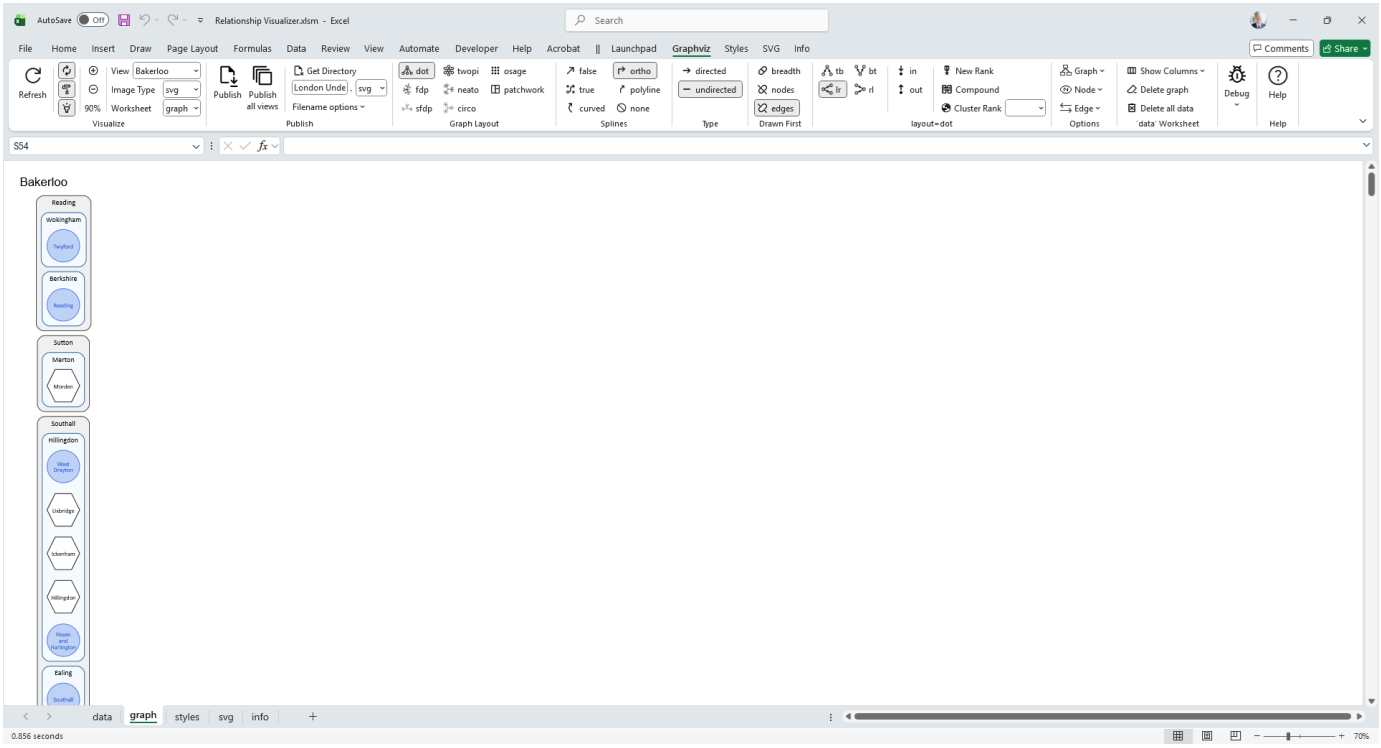


Item	Related Item	Style Name
Baker Street	Regents Park	Bakerloo
Charing Cross	Embankment	Bakerloo
Edgware Road (Bakerloo)	Marylebone	Bakerloo
Embankment	Waterloo	Bakerloo
Harlesden	Willesden Junction	Bakerloo
Harrow and Wealdstone	Kenton	Bakerloo
Kensal Green	Queens Park	Bakerloo
Kenton	South Kenton	Bakerloo
Kilburn Park	Maida Vale	Bakerloo
Lambeth North	Elephant and Castle	Bakerloo
Maida Vale	Warwick Avenue	Bakerloo
Marylebone	Baker Street	Bakerloo
North Wembley	Wembley Central	Bakerloo
Oxford Circus	Piccadilly Circus	Bakerloo
Paddington	Edgware Road (Bakerloo)	Bakerloo
Piccadilly Circus	Charing Cross	Bakerloo
Queens Park	Kilburn Park	Bakerloo
Regents Park	Oxford Circus	Bakerloo
South Kenton	North Wembley	Bakerloo
Stonebridge Park	Harlesden	Bakerloo
Warwick Avenue	Paddington	Bakerloo
Waterloo	Lambeth North	Bakerloo
Wembley Central	Stonebridge Park	Bakerloo
Willesden Junction	Kensal Green	Bakerloo
Bank	St. Pauls	Central
Barkingside	Fairlop	Central
Bethnal Green	Liverpool Street	Central
Bond Street	Marble Arch	Central
Buckhurst Hill	Woodford	Central
Chancery Lane	Holborn	Central
Chigwell	Roding Valley	Central
Debden	Loughton	Central
East Acton	North Acton	Central
Epping	Theydon Bois	Central
Fairlop	Hainault	Central

Step 5 — Notice that **Bakerloo** now appears as a value in the dropdown list. The selections in this list are refreshed automatically whenever a new View column is added to the `styles` worksheet.

Change the selected `View` from **All** to **Bakerloo** on the `Graphviz` ribbon tab.

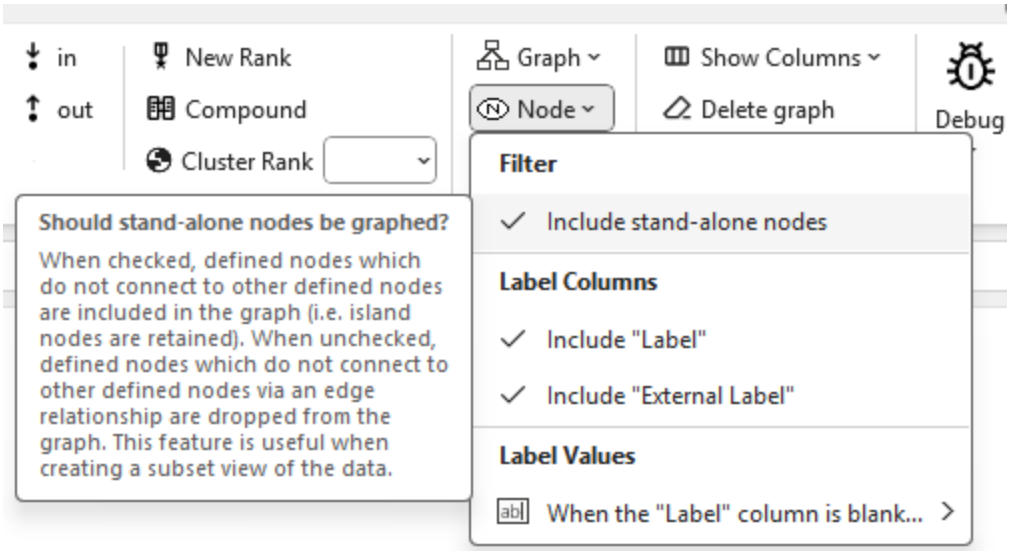
Step 6 — Press the `Refresh` button. The new graph for the Bakerloo view does **not** appear as we might expect. All stations are still visible, and most are no longer connected to anything. What gives?



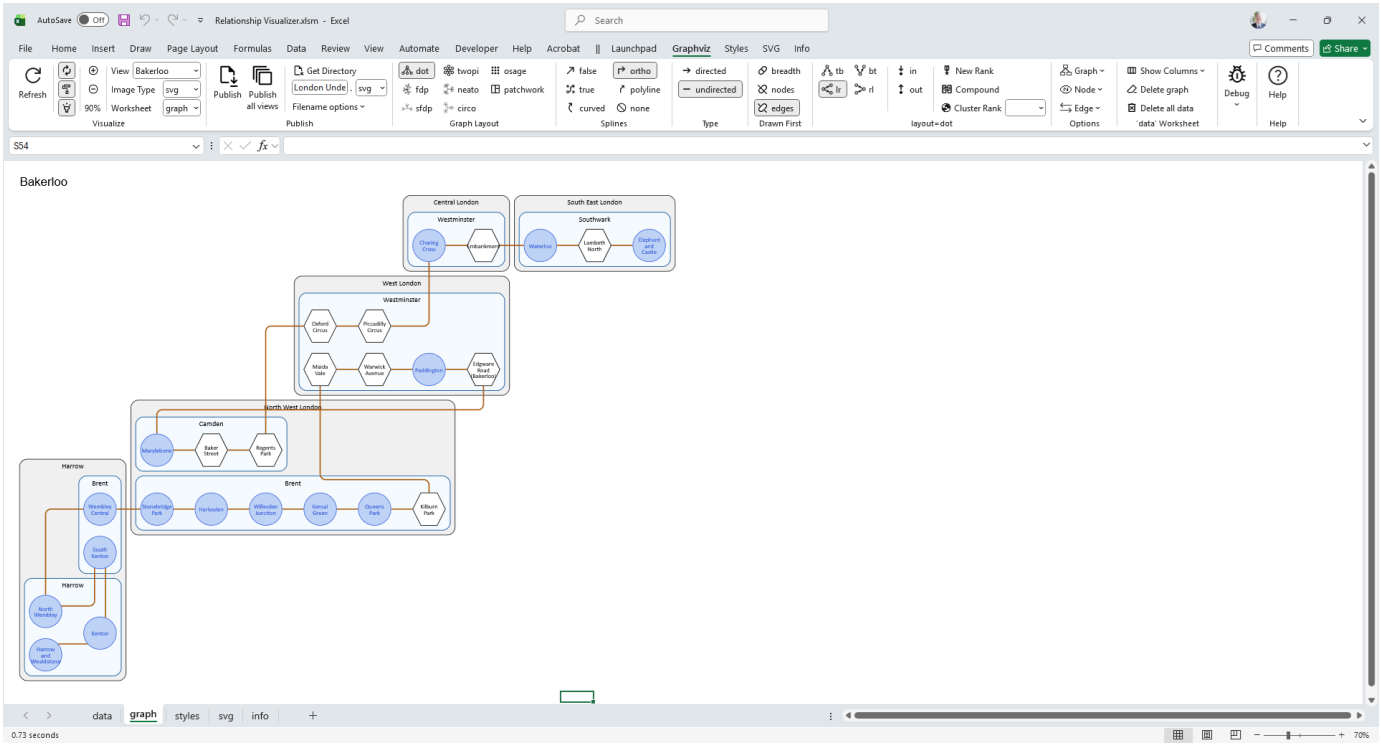
The reason is that we filtered the **edges**, but we did not filter the **nodes**. Every station still has a node style set to **Yes** in the selected View column, so Graphviz dutifully draws all of them. However, because only the Bakerloo edges remain enabled, most stations no longer have any connecting relationships. The result is a graph full of isolated nodes. To fix this, we must also restrict the node styles so that only the stations belonging to the Bakerloo line are included in the view.

The Relationship Visualizer includes switches that can automatically remove these “island” nodes when they serve no purpose in the graph. When enabled, this option hides any node that has no incoming or outgoing edges in the selected view, ensuring that the resulting graph contains only the stations and relationships relevant to the Bakerloo line.

Step 7 - Remove the check mark on the `Graphviz` ribbon from the `Nodes` - `Include stand-alone nodes` switch control. This means only include nodes that have an edge connection to another node.



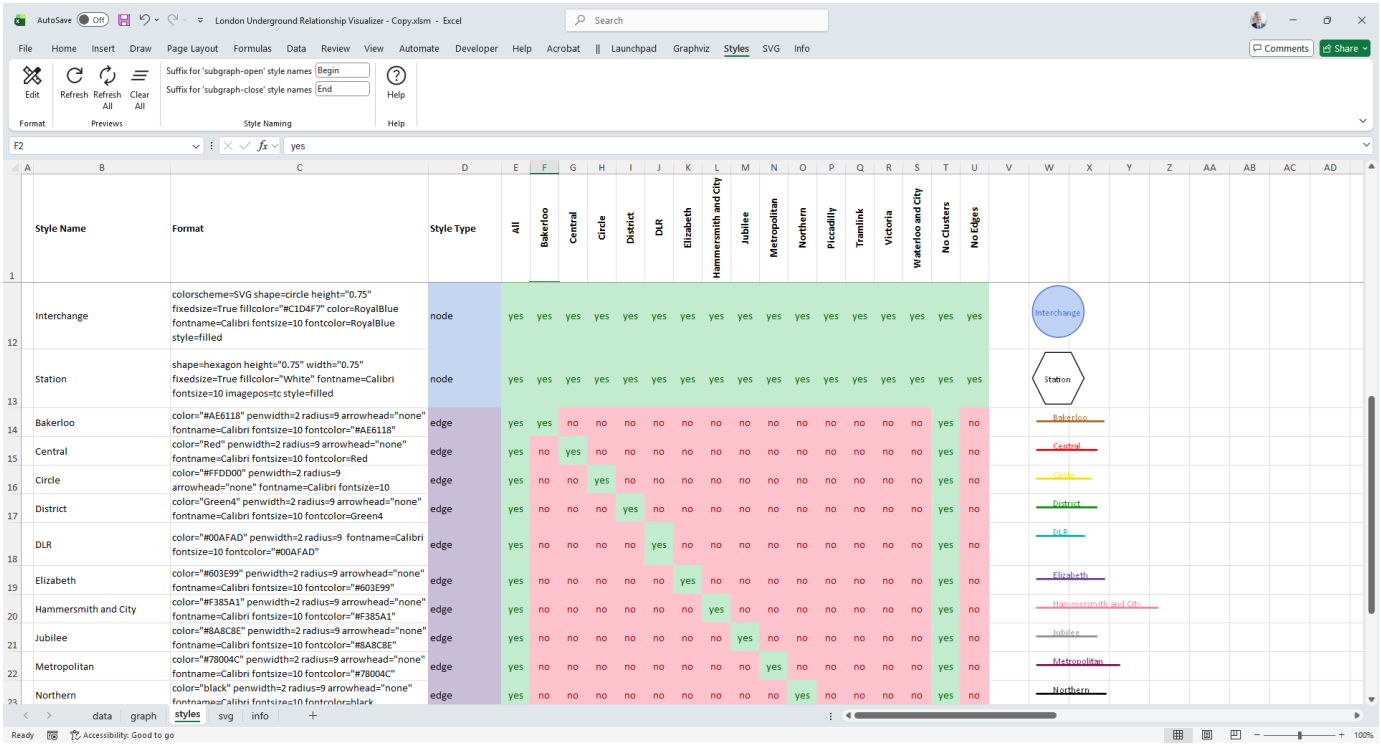
Step 8 — Remove the check mark on the `Graphviz` ribbon from the `Nodes` → `Include stand-alone nodes` switch. This tells the Relationship Visualizer to include only those nodes that have at least one edge connection to another node.



Notice that the various clusters have disappeared along with the stations and interchanges that do not participate in a Bakerloo-style relationship. This happens because the Relationship Visualizer is now excluding stand-alone nodes, and Graphviz performs its own additional filtering: it will not draw a cluster that contains no nodes. The result is a clean graph showing only the stations and connections that belong to the Bakerloo line.

> **Note:** This behavior differs from tools like Microsoft Visio, where layers simply hide or show shapes without affecting their layout or grouping. In Graphviz, filtering out nodes or edges can also remove entire clusters, because clusters only exist if they contain at least one visible node. As a result, the structure of the graph can change when a view is applied.

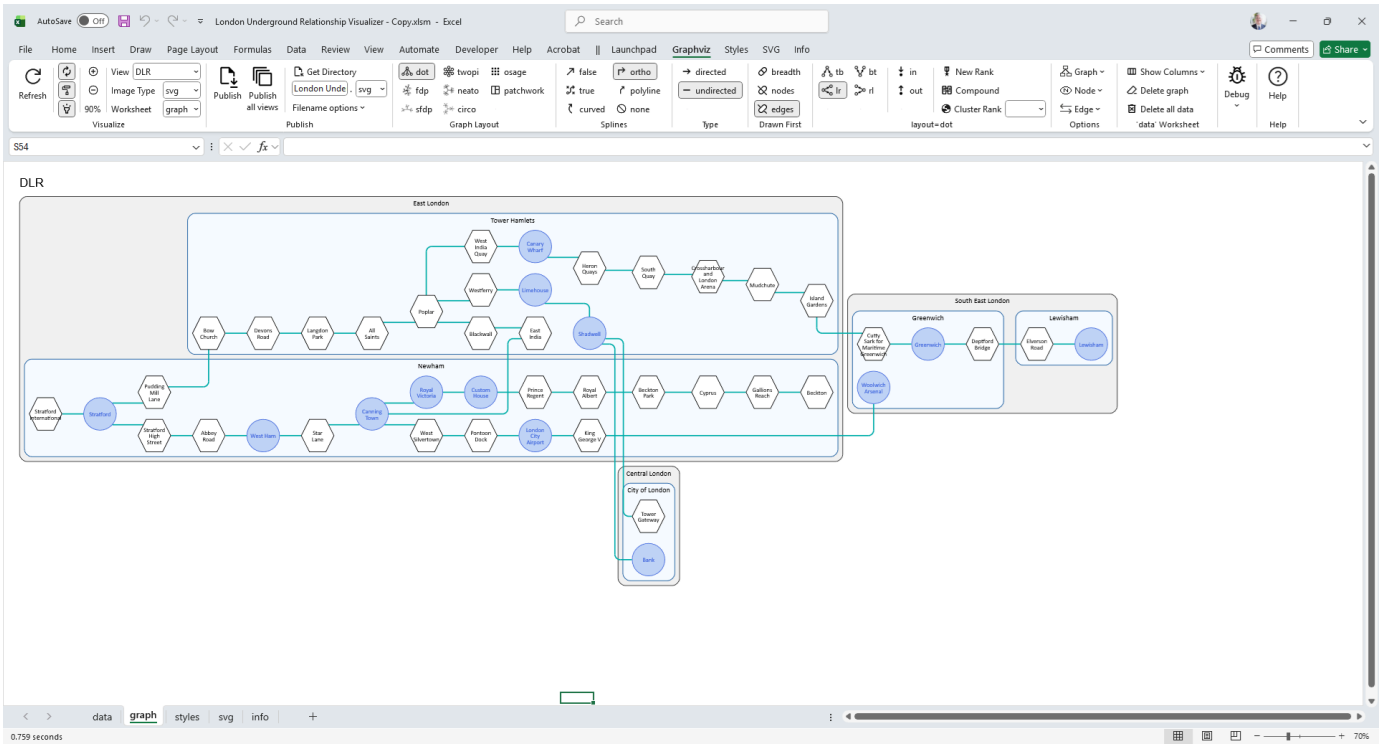
Step 9 — Repeat Steps 2 and 3 for all the remaining Underground lines. The `styles` worksheet should now show a diagonal pattern of **Yes** values across the View columns, and will appear similar to the following:



	Style Name	Format	Style Type	All	Bakerloo	Central	Circle	District	DLR	Elizabeth	Hammersmith and City	Jubilee	Metropolitan	Northern	Piccadilly	Tramlink	Victoria	Waterloo and City	No Clusters	No Edges
12	Interchange	colorscheme=SVG shape=circle height="0.75" fixedsize=True fillcolor="#C1D4F7" color=RoyalBlue fontname=Calibri fontsize=10 fontcolor=RoyalBlue style=filled	node	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
13	Station	shape=hexagon height="0.75" width="0.75" fixedsize=True fillcolor="White" fontname=Calibri fontsize=10 imagepos=tc style=filled	node	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
14	Bakerloo	color="#AE6118" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor="#AE6118"	edge	yes	yes	no	no	no	no	no	no	no	no	no	no	no	no	no	yes	no
15	Central	color="Red" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor=Red	edge	yes	no	yes	no	no	no	no	no	no	no	no	no	no	no	no	yes	no
16	Circle	color="#FFDD00" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10	edge	yes	no	no	yes	no	no	no	no	no	no	no	no	no	no	no	yes	no
17	District	color="Green4" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor=Green4	edge	yes	no	no	no	yes	no	no	no	no	no	no	no	no	no	no	yes	no
18	DLR	color="#00AFAD" penwidth=2 radius=9 fontname=Calibri fontsize=10 fontcolor="#00AFAD"	edge	yes	no	no	no	no	yes	no	no	no	no	no	no	no	no	no	yes	no
19	Elizabeth	color="#603E99" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor="#603E99"	edge	yes	no	no	no	no	no	yes	no	no	no	no	no	no	no	no	yes	no
20	Hammersmith and City	color="#F385A1" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor="#F385A1"	edge	yes	no	no	no	no	no	no	yes	no	no	no	no	no	no	no	yes	no
21	Jubilee	color="#8A8C8E" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor="#8A8C8E"	edge	yes	no	no	no	no	no	no	no	yes	no	no	no	no	no	no	yes	no
22	Metropolitan	color="#78004C" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor="#78004C"	edge	yes	no	no	no	no	no	no	no	no	yes	no	no	no	no	no	yes	no
23	Northern	color="black" penwidth=2 radius=9 arrowhead="none" fontname=Calibri fontsize=10 fontcolor=black	edge	yes	no	no	no	no	no	no	no	no	no	yes	no	no	no	no	yes	no

The list of available views on the Graphviz tab expands automatically to match the new column headings. It will appear as:

Step 10 — Pick another subway line, such as **DLR**. The graph updates to:



You now have the ability to switch between subway lines using a single shared data set. You can also use the `Publish all views` button to quickly generate one output file per view, allowing you to produce a complete set of line-specific graphs with a single command.

With these views in place, the Relationship Visualizer becomes a powerful exploration tool. A single shared data set can now produce a family of focused, line-specific graphs, each generated on demand or published in bulk. Whether you are analyzing connectivity, documenting infrastructure, or simply exploring the network, Views give you a flexible, repeatable way to highlight exactly the relationships you care about.

Publishing Graphs

One of the strengths of the Relationship Visualizer is its ability to handle large datasets and let Graphviz determine an efficient layout automatically. However, graphs with substantial amounts of data can become quite large—often far beyond what can be comfortably viewed within Excel.

Publishing the graph to an external file is often the most practical way to work with these larger diagrams. Saving the output allows you to archive versions of the graph as your data evolves, share the results with colleagues who may not be using Excel, and review the structure of the graph independently of the workbook.

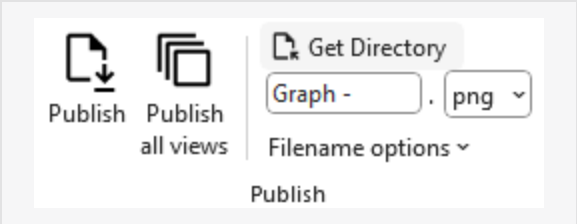
If you need to examine these diagrams in detail or print them, you will likely want to use a dedicated viewer such as **Adobe Acrobat Reader**. Tools like this allow you to zoom in and out smoothly and use poster-printing features to span the diagram across multiple sheets of paper.

When to Publish Instead of Refresh

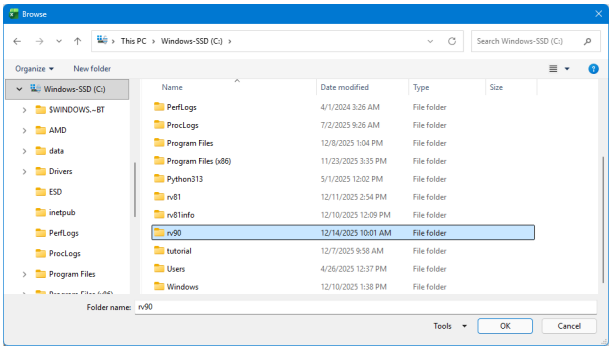
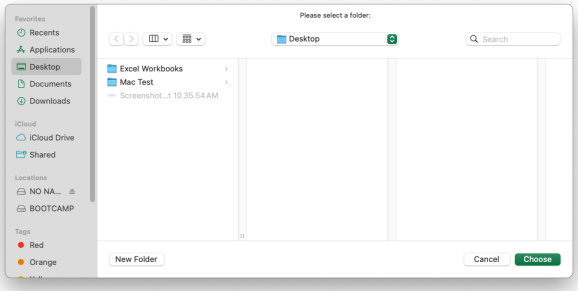
Use **Publish** when you want to work with the graph outside of Excel—whether to archive versions, share the diagram with others, or explore the layout at a scale Excel cannot comfortably display. Publishing is also ideal when you are comparing multiple layout engines or spline settings, since each exported file can serve as a snapshot of your experiments. If you plan to print the diagram or examine fine details, publishing to a PDF or SVG and opening it in a dedicated viewer will give you far more flexibility than the in-workbook preview.

Setting Output File Options

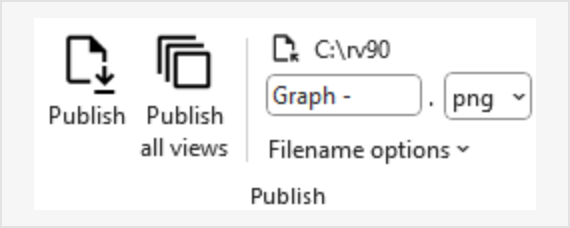
You must specify a directory where you want graph files written to, and provide a filename prefix for the file. Select the `Get Directory` button in the `Publish` group on the `Graphviz` ribbon tab.



A directory-picking dialog will appear:

Windows	macOS
	

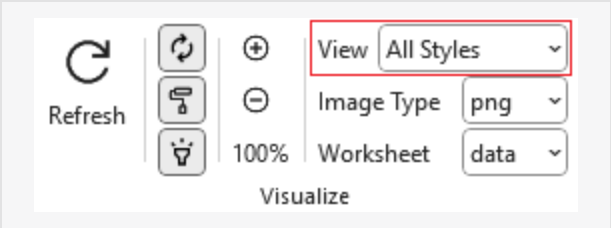
Choose a directory, press OK, and the directory path replaces the `Get Directory` text on the selection button.



Next, specify a filename prefix in the edit field below the directory name. This is a value that the filename will begin with. Here the file name prefix is set to `Graph -`.

You control the type of file generated by specifying the extension after the prefix. In this example will are creating a `png` image.

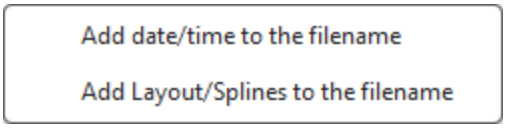
The heading value of the view column used to control which styles were included in the diagram will be appended to the prefix as part of the file name. In this example, we will use the `All Styles` view.



A dropdown menu below the file prefix provides additional switches to append a date and time to the filename and to include the graph options used (layout engine and spline setting) when generating the graph.

- Adding a timestamp helps ensure that new files can be created during iterative refinement. Graphviz cannot overwrite a file that is currently open in another application, such as Acrobat Reader, so unique filenames prevent interruptions.
- Including the graph options in the filename allows you to experiment with different layout engines and spline settings and easily identify which combination produced the most effective result.

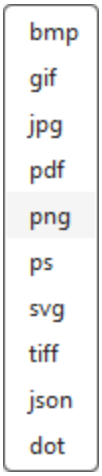
Storing this information directly in the filename provides a convenient way to recall the settings you selected once you have settled on the configuration that works best.



In this example, all the options have been enabled.

Specifying the Output File Format

Graphviz provides numerous file formats that the diagrams can be written as, such as **gif**, **jpeg**, **pdf**, or **tiff**. The Relationship Visualizer provides the most commonly used file formats in the `File Format` dropdown list.



Publish / Publish all views

Press the `Publish` button.

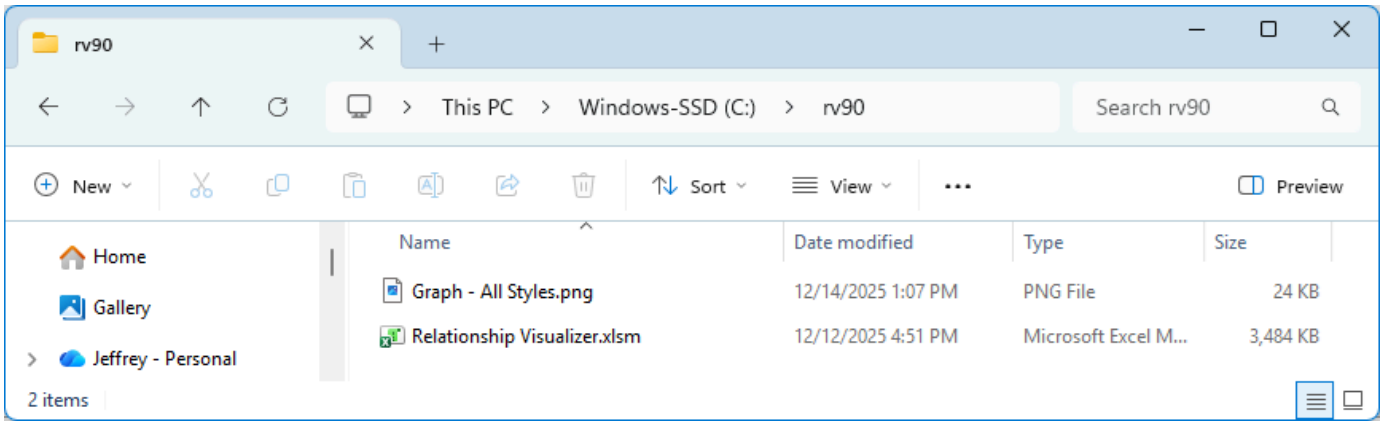
A graph is generated in the same manner as when `Refresh` is pressed; however, it is not displayed in the Excel workbook. Instead, a message appears in the Excel status bar indicating the name of the file that was created and the folder where it was saved.

You may also choose to have one file per view created by selecting the `Publish all views` button.

The list of views will be iterated in a loop, and a file will be published based on the yes/no switches specified for that view. The View Name will be appended to the file prefix allowing you to tell the graphs apart.

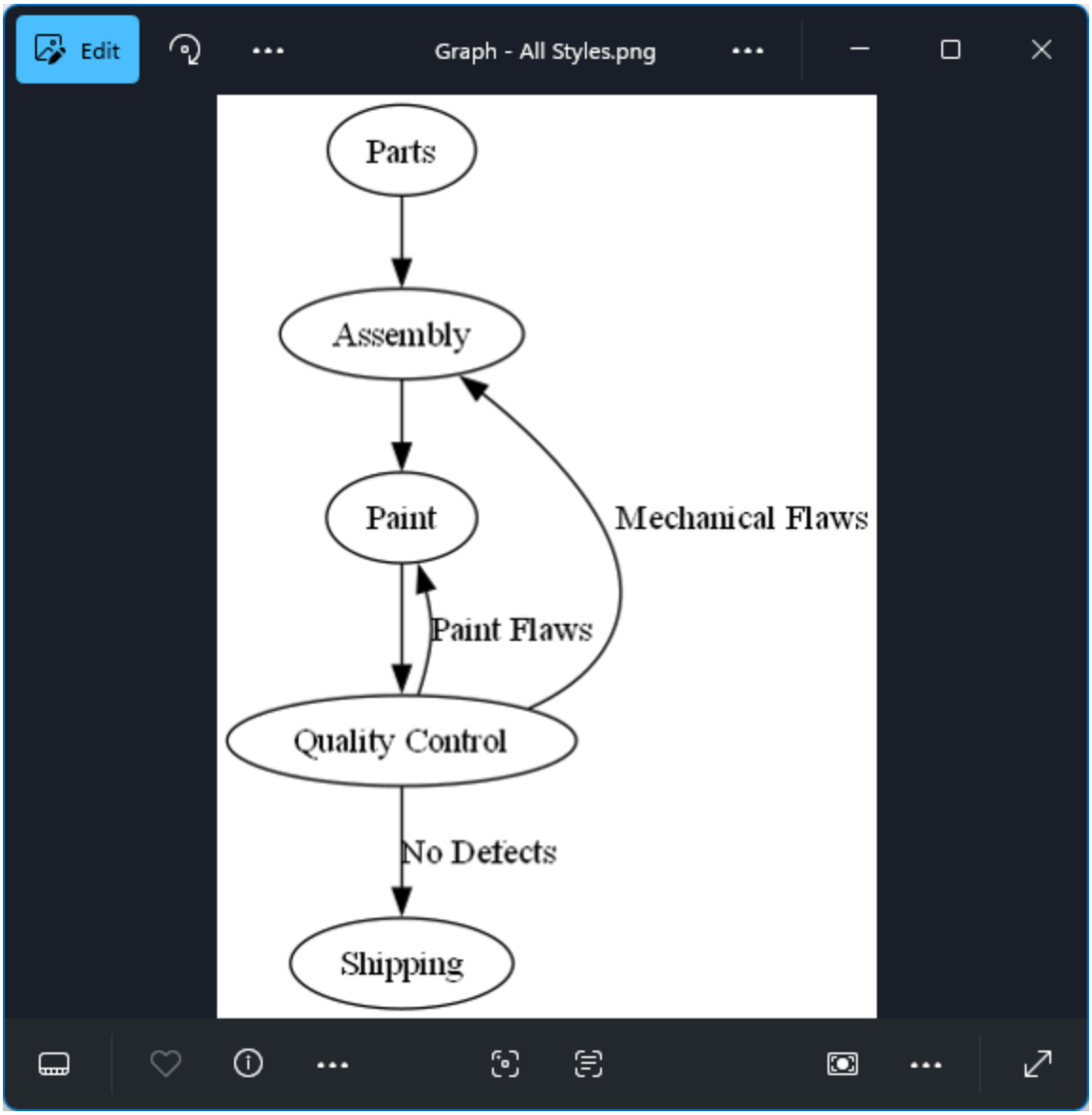
View the File

Bring up Windows Explorer and find your file. Note that the file name is a concatenation of the prefix (`Graph -`) and the View Name (`All Styles`) and the file extension (`.png`).



Launch the file, and see it displayed.

The application associated with the file type will display the file. In this example, the graph is a `png` file, which is associated with Windows image viewer. All editing, zoom, print, and annotation capabilities it provides are available to you.


Edit
Graph - All Styles.png
Parts
Assembly
Paint
Mechanical Flaws
Paint Flaws
Quality Control
No Defects
Shipping

Post-Process SVG Files

Graphviz can generate SVG output using the `Publish` and `Publish all views` features.

Relationship Visualizer extends this by allowing you to post-process the generated SVG. Using simple find-and-replace rules, you can modify the SVG's XML to add styling, adjust structure, or inject JavaScript for animations and interactive behaviors.

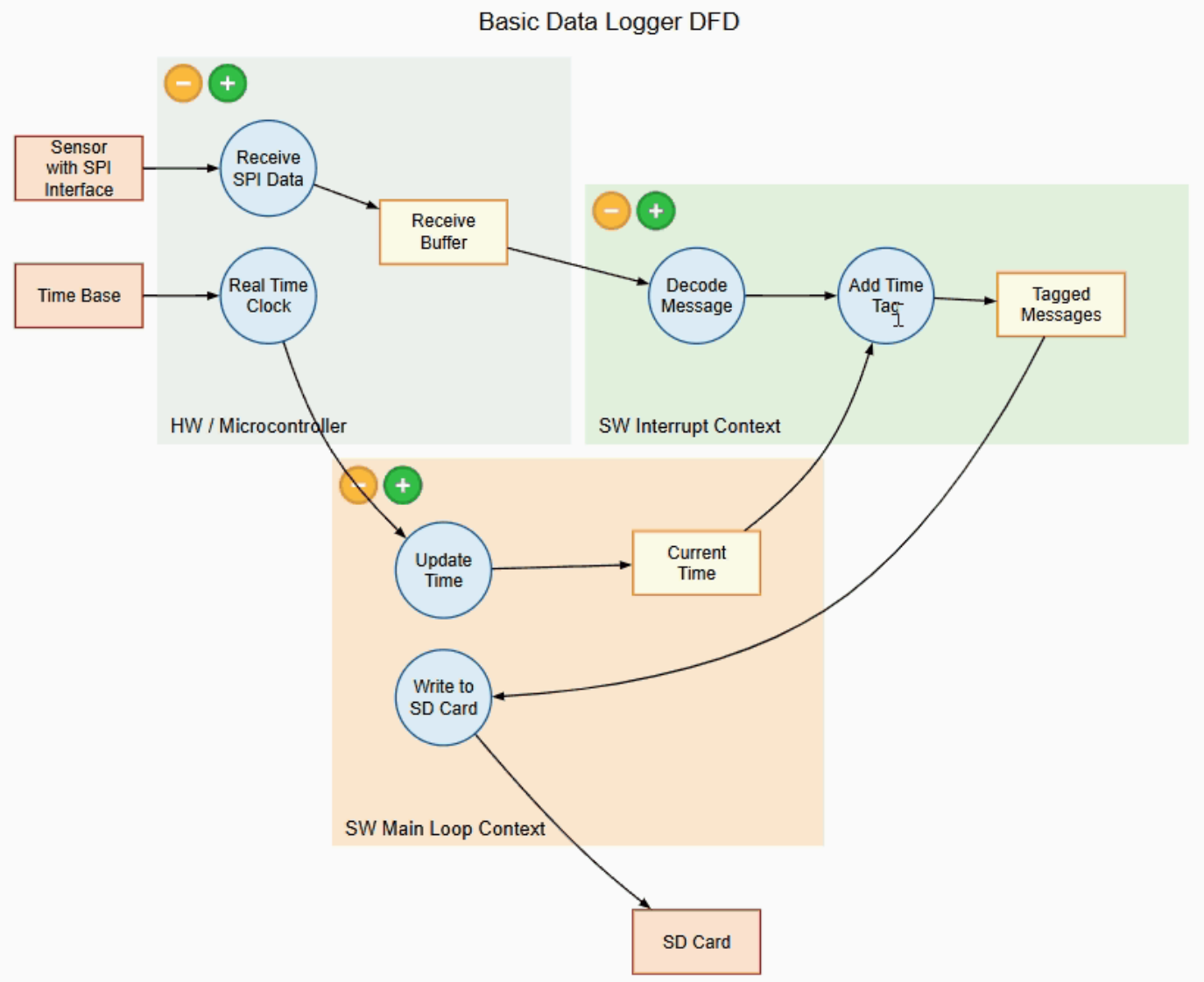
What is SVG?

SVG stands for *Scalable Vector Graphics*, an XML-based format for describing two-dimensional vector graphics. SVG images scale infinitely without losing quality, making them ideal for diagrams, illustrations, and web-based visuals. They are lightweight, text-based, and support interactivity and animation.

What is Post-processing?

Post-processing refers to the transformations applied to an SVG *after* Graphviz has finished generating it. While Graphviz produces a structurally correct diagram, the raw SVG often benefits from additional cleanup or enhancement.

The animated GIF below shows a Graphviz-generated SVG with post-processing applied to add animation.



Post-processing allows you to apply additional styling to nodes, edges, polygons, and polylines. It also enables the insertion of JavaScript for interactive behaviors such as click-event animations and zooming. In short, post-processing turns a technically valid SVG into a polished, presentation-ready graphic.

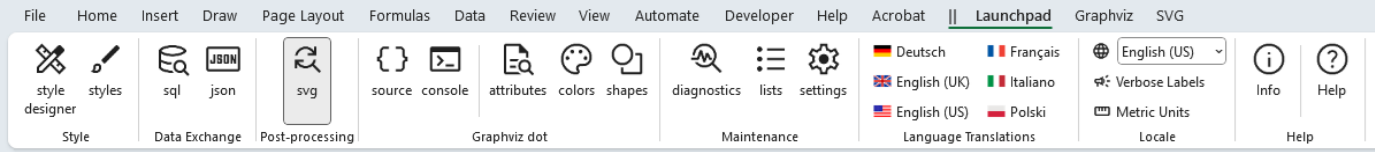
The `svg` worksheet defines a series of **Find** and **Replace** operations, with optional comments. Any row can be commented out by placing a `#` in column A.

When SVG post-processing is active, each newly created SVG file is loaded into memory, and the `svg` worksheet is processed from top to bottom. Each row's **Find** value is searched for in the SVG's XML, and any matches are replaced with the corresponding **Replace** value.

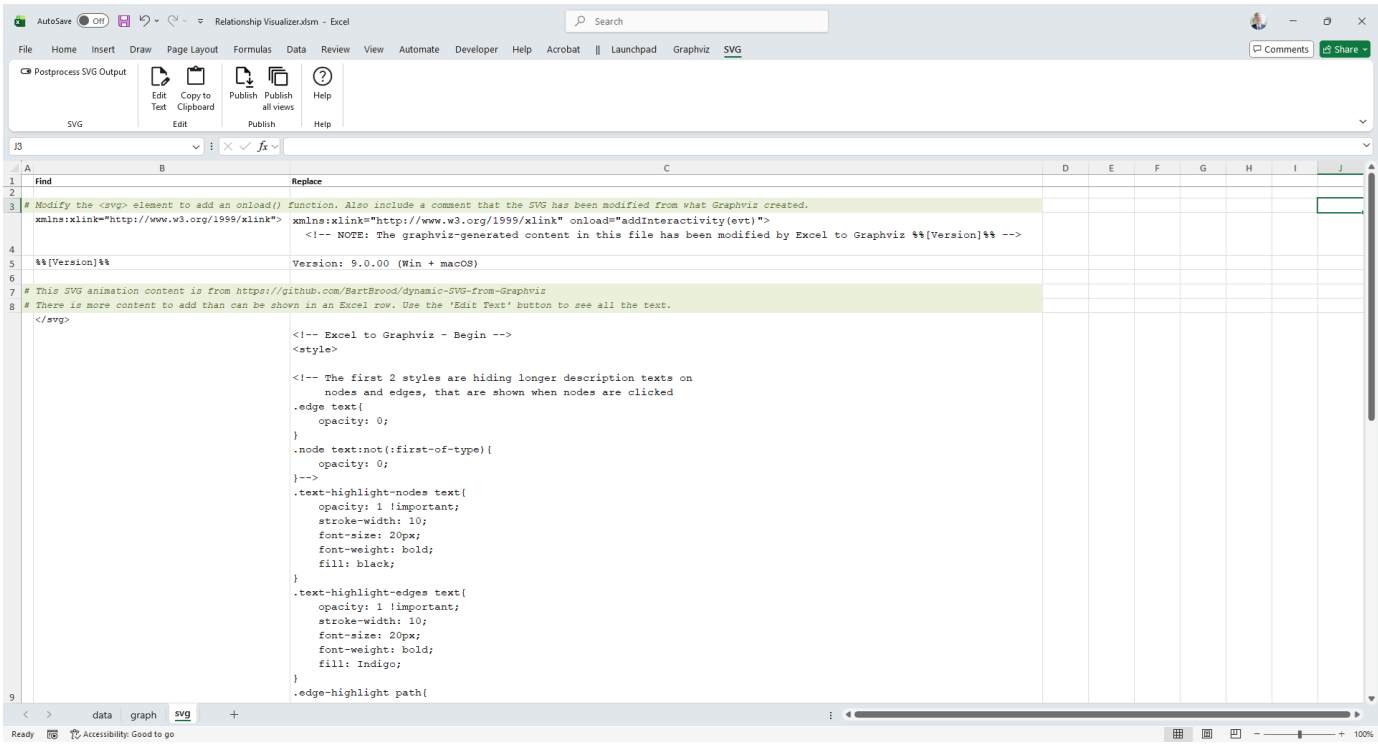
Post-processing is **disabled by default** and must be explicitly enabled.

The `svg` Worksheet

The `svg` worksheet is reached from the `Post-processing` section of the Launchpad ribbon tab.



The default `svg` worksheet appears as follows:



The `svg` Worksheet has the following columns:

A	B	C
Indicator	Find	Replace

The columns are as follows:

(A) Indicator

Allows you to place a `#` character to denote a comment. This can be used to comment out a find/replace pair so it is excluded from the post-processing.

(B) Find

Specifies the string to seach for.

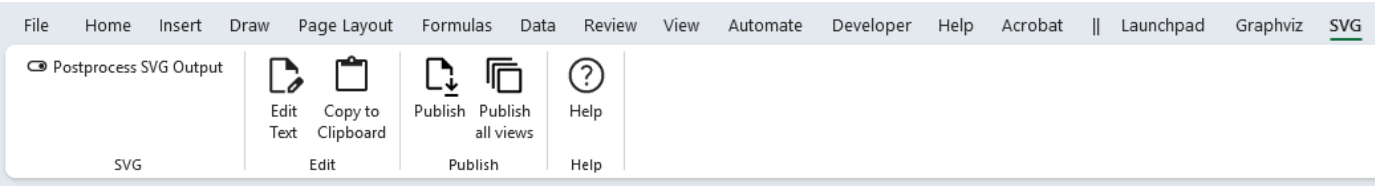
(C) Replace

Contains the string to substitute for the string in column B.

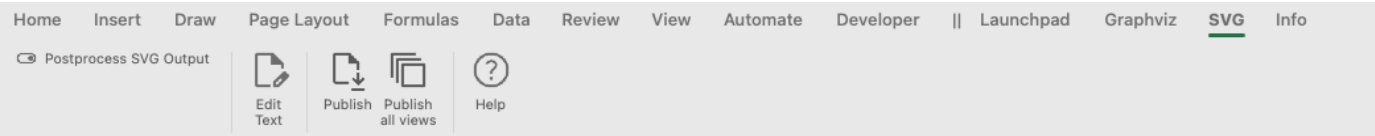
The `SVG` Ribbon Tab

The `SVG` ribbon tab is activated whenever the `svg` worksheet is activated. It appears as follows:

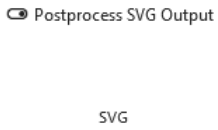
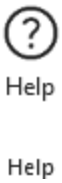
`Windows`



`macOS`



It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
SVG		Controls when post-processing is performed.
Edit		Provides tools to help get around Excel's inability to display large amounts of cell text.
Publish		Provides convenience buttons to invoke the publishing buttons which reside on the `Graphviz` ribbon tab.
Help		Provides a link to the `Help` content for the `svg` worksheet (i.e. this web page).

SVG

Controls when post-processing is performed.

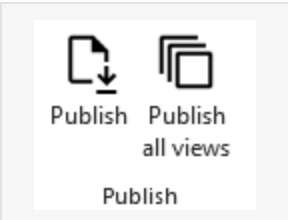
Label	Control Type	Description
Postprocess SVG Output	Checkbox	Disabled by default. When checked, post-processing Find/Replace is performed whenever a graph is written to file in `SVG` format.

Edit

Provides tools to help get around Excel's inability to display large amounts of cell text.

Label	Control Type	Description
Edit Text	Button	Launches the Edit Text form with the contents of the currently selected cell. A second location where the **Edit Text** button appears is as a floating pencil button on the right side of any selected `Replace` cell.  Clicking the pencil button performs the same action as selecting the **Edit Text** button in the Ribbon.
Copy to Clipboard	Button	Copies the contents of the cell as straight text to the Microsoft Windows clipboard, so it can be pasted into an external editor. Characters such as quotes are not escaped as would occur when using Excel's copy (Ctrl+C).

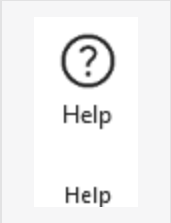
Publish



Provides convenience buttons to invoke the publishing buttons which reside on the `Graphviz` ribbon tab. As you tune your post-processing code, having these buttons eliminates the need to bounce back and forth to the `Graphviz` ribbon tab.

Label	Control Type	Description
Publish	Button	Creates a graph, using the View currently chosen on the `Graphviz` tab worksheet and performs the SVG post-processing on the generated file.
Publish all views	Button	Creates one graph file per view on the styles worksheet and performs the post-processing on each file.

Help



Provides a link to the `Help` content for the `svg` worksheet (i.e. this web page).

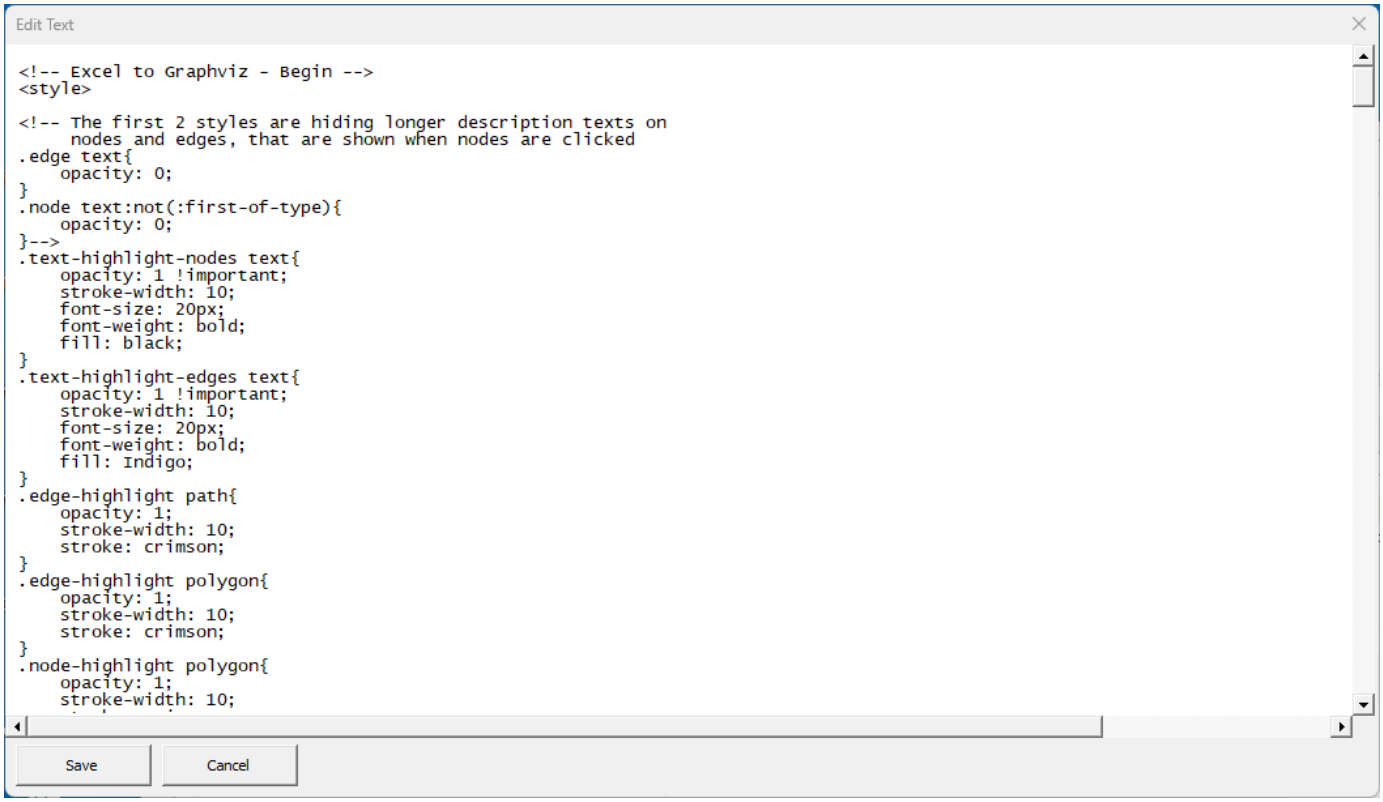
Label	Control Type	Description
Help	Button	Provides a link to this web page.

The `Edit Text` Form

Excel rows can hold a great deal of information, but cells have their quirks. Each cell can store up to 32,767 characters, yet only about 1,024 of them are visible directly in the grid without entering cell-edit mode. For rows, the practical display limit depends on the combined content of each cell, though in reality screen size and readability become limiting

factors long before Excel's technical limits are reached. In short, you can store a lot—but you may not be able to see it all at once.

To address this, Version 7 introduced an `Edit Text` form. When you press the **Edit Text** button, a modal window opens containing the full text of the currently selected cell, making it easier to view and edit long content without fighting Excel's display constraints.



Horizontal and vertical scroll bars are provided to help navigate the text. You can change the text within the form.

Pressing the `Save` button transfers the contents from the form back to the active cell.

Best Practices for SVG Post-processing

Post-processing is a powerful feature, but it works best when the Find/Replace rules are written with care. The following guidelines help ensure reliable, predictable results:

- **Target specific patterns.**
  Use Find values that are precise enough to avoid unintended matches. For example, search for `fill="#000000"` instead of just `fill=` to prevent altering unrelated elements.

- **Avoid overly broad replacements.**
  Replacing short or common strings (such as `id=` or `stroke=`) can lead to accidental changes throughout the SVG. When in doubt, narrow the scope.

- **Use comments to document intent.**
  Add a brief comment in column C explaining why a rule exists. This makes the worksheet easier to maintain and helps future you remember what each rule was meant to accomplish.

- **Disable rules instead of deleting them.**
  Prefix a row with `#` to temporarily disable it. This preserves the rule for later use and makes troubleshooting far easier.

- **Order matters.**
  The worksheet is processed from top to bottom. Place general rules first and more specific overrides later, or vice-versa, depending on the effect you want.

- **Test incrementally.**
  Add or modify one rule at a time, then publish a single SVG to confirm the result. This prevents multiple changes from interacting in unexpected ways.

- **Keep Find/Replace pairs reversible when possible.**
  If you may need to undo a transformation later, choose patterns that can be cleanly reversed or re-applied.

- **Be mindful of JavaScript injection.**
  When adding scripts for animation or interactivity, ensure the inserted code is self-contained and does not rely on external libraries unless you explicitly include them.

- **Use unique markers for custom elements.**
  If you add custom classes, IDs, or attributes, prefix them with something distinctive (e.g., `rv-zoom-`, `rv-animate-`) to avoid collisions with Graphviz-generated names.

- **Validate the final SVG.**
  After post-processing, open the SVG in a browser or viewer to confirm that the XML remains well-formed and that the intended visual changes appear correctly.

Advanced Graphviz Topics

This section describes several miscellaneous Graphviz features that can be used to create more elaborate graphs.

HTML-like Labels

Graphviz has a feature where if the value of a label attribute for nodes, edges, clusters, or graphs is given as an HTML string that is delimited by `<` ... `>`, the label is interpreted as an HTML description. At their simplest, such labels can describe multiple lines of variously aligned text as provided by ordinary string labels. More generally, the label can specify a table like those provided by HTML, with different graphical attributes at each level.

The features and syntax supported by these labels are modeled on HTML. However, there are many aspects that are relevant to Graphviz labels that are not in HTML and, conversely, HTML allows various constructs which are meaningless in Graphviz. The Graphviz creators generally refer to these labels as `HTML-Like Labels` but the reader is warned that these labels are not HTML.

The grammar which Graphviz will accept is fully described at:
https://www.graphviz.org/doc/info/shapes.html#html ↗

A basic HTML label can be constructed as text wrapped in the `<` and `>` delimiters as described above.

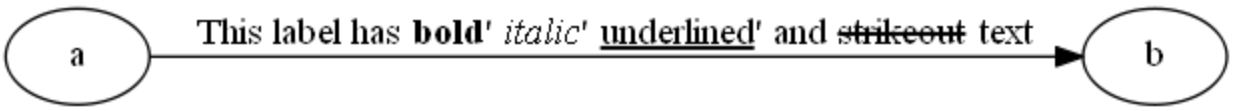
For example, a label can be constructed as:

```
<This label has <b>Bold</b>, <i>italic</i>, <u>underlined</u>, and <s>strikeout</s> text>
```

and entered as a label value for an edge. In this example, we will relate 'a' to 'b' as we are interested in seeing how the edge is drawn. The 'data' worksheet appears as:

	A	B	D	H
1		**Item**	**Label**	**Related Item**
2				
3		a	<This label has Bold, italic, <u>underlined</u>, and <s>strikeout</s> text>	b

Pressing `Refresh Graph` produces the following graph:



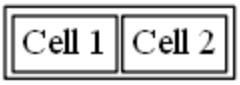
A slightly more complex example is to create a HTML table. In this example, the table contains one row with two cells:

```
<
<table>
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
>
```

Using it to represent a node named 'c', the 'data' worksheet appears as:

	A	B	D	H
1		**Item**	**Label**	**Related Item**
2				
3		a	< <table> <tr> <td>Cell 1</td> <td>Cell 2</td> </tr> </table> >	

Pressing `Refresh Graph` produces the following graph:

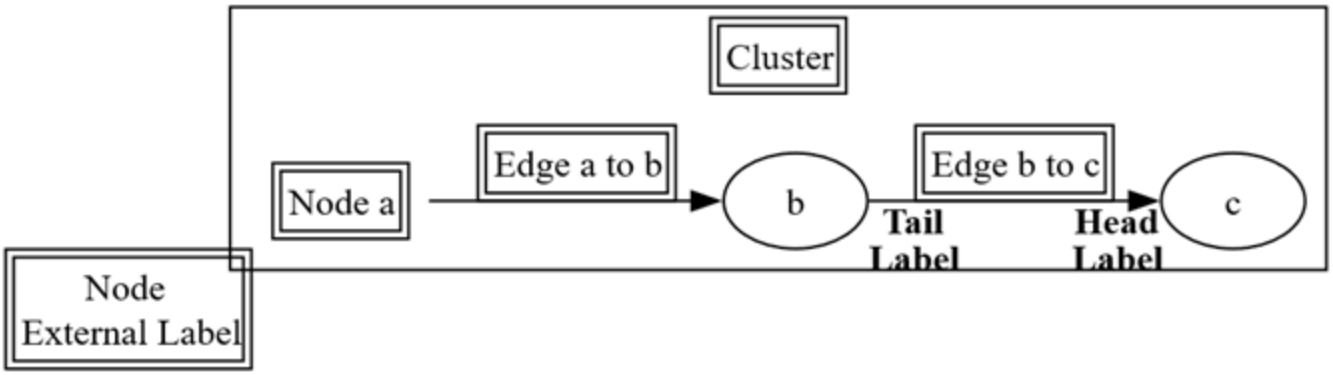


HTML labels can be used for Clusters, Nodes, and Edges. In the example below there are three Items named 'a', 'b', and 'c'. HTML labels have been added for node 'a', and the edges from 'a' to 'b' and from 'b' to 'c'. The nodes and edges are wrapped with a border via a cluster that also has an HTML label.

The 'data' worksheet appears as:

	A	B	C	D	E	F
1		**Item**	**Tail Label**	**Label**	**External Label**	**Head Label**
2		{		< <table> <tr> <td>Cluster</td> </tr> </table> >		
3		a		< <table> <tr> <td>Node a</td> </tr> </table> >	< <table> <tr> <td>Node External Label</td> </tr> </table> >	
4		a		< <table> <tr> <td>Edge a to b</td> </tr> </table> >		
5		b	<Tail Label>	< <table> <tr> <td>Edge b to c</td> </tr> </table> >		<Head Label>

Pressing `Refresh Graph` produces the following graph:



`graph`, `node`, & `edge` Keywords

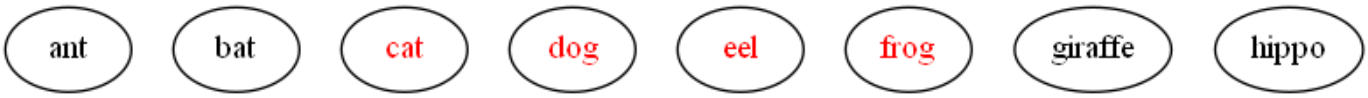
Graphviz supports a cascading default-attribute mechanism: when a `node`, `edge`, or `graph` statement defines a default attribute, every subsequently defined object of that type automatically inherits the value. The default remains in effect until it is reassigned. Objects created *before* a default is introduced do not retroactively adopt it; instead, Graphviz records an empty string for that attribute on those earlier objects.

The Relationship Visualizer mirrors this behavior by treating the keywords `graph`, `node`, and `edge` as special values in the **Item** column of the **data** worksheet. When these keywords appear, the tool inserts the corresponding default-attribute statements into the generated DOT source, applying the styles specified in the **Style Name** and **Attributes** columns (when those columns are enabled on the *settings* worksheet).

In the example below, nodes `a` through `h` are listed normally. On row 5, a `node` statement sets the default font color to red. Because this row is visually distinguished through conditional formatting (highlighted background and bold-italic text), it is easy to spot as a default-attribute definition. All nodes defined after row 5 inherit the red font. This continues until row 10, where a second `node` statement resets the font color to an empty string, instructing Graphviz to revert to its built-in default.

	A	B	D	H	I	J
1		**Item**	**Label**	**Related Item**	**Style Name**	**Attributes**
2						
3		a	ant			
4		b	bat			
5		***node***				fontcolor=red
6		c	cat			
7		d	dog			
8		e	eel			
9		f	frog			
10		***node***				fontcolor=""
11		g	giraffe			
12		h	hippo			

Pressing `Refresh Graph` produces the following graph:



Likewise, this same capability exists for edges using the `edge` keyword. In the example below an edge keyword on row 13 sets the edge color to blue for the first 3 edges. A second edge keyword on row 17 changes the color to red for all the remaining edges.

	A	B	D	H	I	J
1		**Item**	**Label**	**Related Item**	**Style Name**	**Attributes**
2						
3		a	ant			
4		b	bat			
5		***node***				fontcolor=red
6		c	cat			
7		d	dog			
8		e	eel			
9		f	frog			
10		***node***				fontcolor=""
11		g	giraffe			
12		h	hippo			
13		***edge***				color=blue
14		a		b		
15		b		c		
16		c		d		
17		***edge***				color=red
18		d		e		
19		e		f		
20		f		g		
21		g		h		

Produces the following graph:

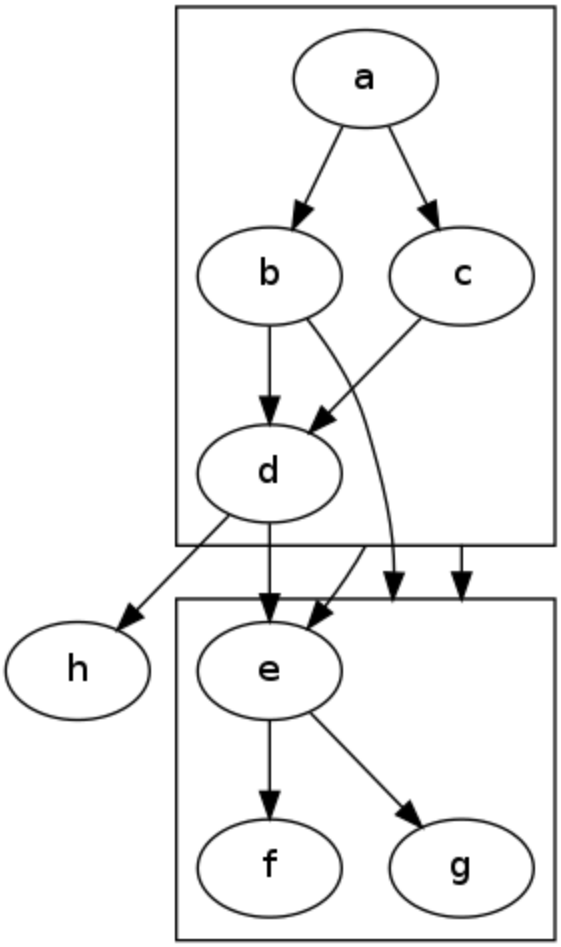
Note that a subgraph receives the attribute settings of its parent graph at the time of its definition. This can be useful; for example, one can assign a font to the root graph and all subgraphs will also use the font. For some attributes, however, this property is undesirable. If one attaches a label to the root graph, it is probably not the desired effect to have the label used by all subgraphs. Rather than listing the graph attribute at the top of the graph, and the resetting the attribute as needed in the subgraphs, one can simply defer the attribute definition in the graph until the appropriate subgraphs have been defined.

Edges from or to a Cluster

You may encounter situations where your diagram includes nodes inside clusters, and you need to show dependencies between those nodes and other nodes or clusters. In other

words, you want an edge to start or end on the boundary of a cluster rather than on a specific node. The Relationship Visualizer supports this, but it requires a small amount of additional Graphviz knowledge to enable cluster-to-cluster or node-to-cluster connections.

Let's reproduce the diagram below which can be found at:
http://stackoverflow.com/questions/2012036/graphviz-how-to-connect-subgraphs ↗



Let's begin by turning the previously described `graph`, `node`, and `edge` keyword features into a concrete example. In rows 4–6 below, each keyword is listed along with style settings that apply to that object type. These statements are not required for connecting clusters, but they demonstrate how the keyword mechanism works and make the resulting diagram easier to read. For each of the three object types—graph, node, and edge—enter the following formatting information into the corresponding `Attributes` cells:

- *Graph*: `fontname="Arial" fontsize="12" fontcolor="red"`
- *Node*: `fontname="Arial"`

- *Edge*: `fontname="Arial" fontsize="8" decorate="true" color="blue"`

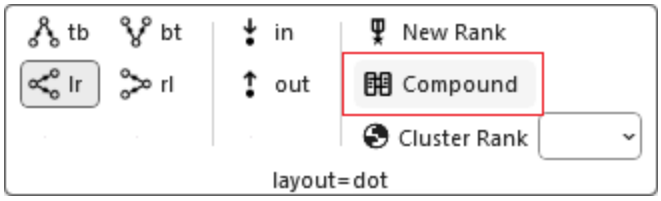
The spreadsheet should look as follows:

	A	B	C	G	H	I
1		**Item**	**Label**	**Is Related To Item**	**Style Name**	**Extra Style Attributes**
2	#	*Set the font and font size for clusters, nodes, and edges*				
3		***graph***				fontname="Arial" fontsize="12" fontcolor="Red"
4		***node***				fontname="Arial"
5		***edge***				fontname="Arial" fontsize="8" color="Blue" decorate="true"

Next, we need to enable the `compound` graph attribute, which Graphviz sets to `false` by default. Setting it to `true` activates support for edges that connect to clusters. The simplest way to do this is to follow the steps described in Adding Native Graphviz Directives and insert a native Graphviz statement. Place a `>` character in the **Item** column to mark the row as a native command, and enter `compound="true"` in the **Label** column, as shown below.

6	#	*Enable 'compound' graph attribute*				
7		>	compound="true"			

The `compound="true"` statement is then added to the body of the main graph. You can achieve the same result by pressing the **Compound** option button, which sets this attribute automatically.



Next, define two clusters. For clarity, we'll label them `cluster0` and `cluster1`. `cluster0` will contain four node relationships using the letters **a**, **b**, **c**, and **d**, all of which will appear inside its cluster boundary.
Similarly, `cluster1` will contain node relationships using **e**, **f**, and **g**, also enclosed within a cluster border.

Up to this point, we have always started a cluster with an opening brace `{`. When the Relationship Visualizer encounters an opening brace, it automatically generates an internal

name for the cluster to simplify authoring. However, if you want to connect edges *to* a cluster, that cluster must have a stable Item name. Relying on the auto-generated name can be unreliable, because it may change if the cluster's position in the worksheet changes.

To address this, the Relationship Visualizer allows you to place a name *before* the opening brace (for example, `cluster0`) to explicitly define a named subgraph. If the name begins with **cluster**, Graphviz treats the subgraph as a special cluster subgraph. When supported by the layout engine, nodes in that cluster are grouped together and enclosed within a bounding rectangle. Note that cluster subgraphs are not part of the formal DOT language—they are a syntactic convention recognized by certain Graphviz layout engines. If the name does *not* begin with **cluster**, the subgraph is treated as an ordinary subgraph.

The spreadsheet now appears as follows to define the two clusters. In the illustration, notice how the cluster name appears in both the **Item** and **Label** columns. We will remove the label later in this example, but for now it helps make the demonstration clearer.

8	#	*cluster0*				
9		cluster0 {	cluster0			
10		a		b		
11		a		c		
12		b		d		
13		c		d		
14		}				
15	#	*cluster1*				
16		cluster1 {	cluster1			
17		e		g		
18		e		f		
19		}				

Once the clusters have been defined, we can specify edges that illustrate the relationships between them. The example below demonstrates five scenarios:

1. A relationship from a node inside one cluster to a node inside another cluster (row 21)
2. A relationship from a node inside a cluster to a node outside any cluster (row 22)
3. A relationship from a node inside a cluster to the *border* of another cluster (row 23)
4. A relationship from the *border* of a cluster to a node inside another cluster (row 24)
5. A relationship from the border of one cluster to the border of another cluster (row 25)

To specify the cluster from which an edge should originate, include an `ltail` attribute in the **Attributes** column. This attribute identifies the cluster whose boundary the edge should

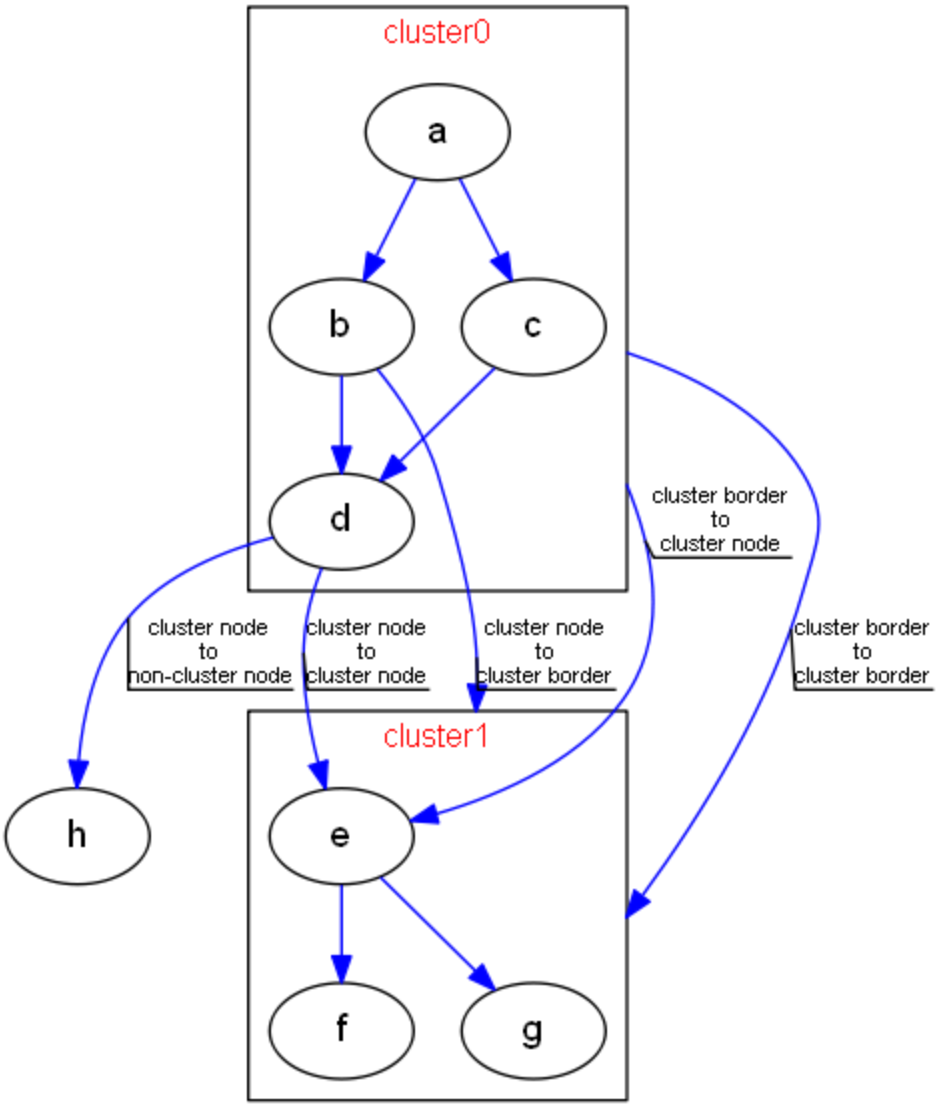
start from—for example, `ltail="cluster0"`. The item listed in the **Item Name** column must belong to that cluster.

To make an edge terminate at the boundary of a cluster, include an `lhead` attribute in the **Attributes** column. This identifies the cluster whose border the arrowhead should connect to—for example, `lhead="cluster1"`. The item listed in the **Related Item** column must reside within that cluster.

The spreadsheet below shows all five scenarios, with descriptive text added in the **Label** column to clarify each case. These labels will be hidden later in the example.

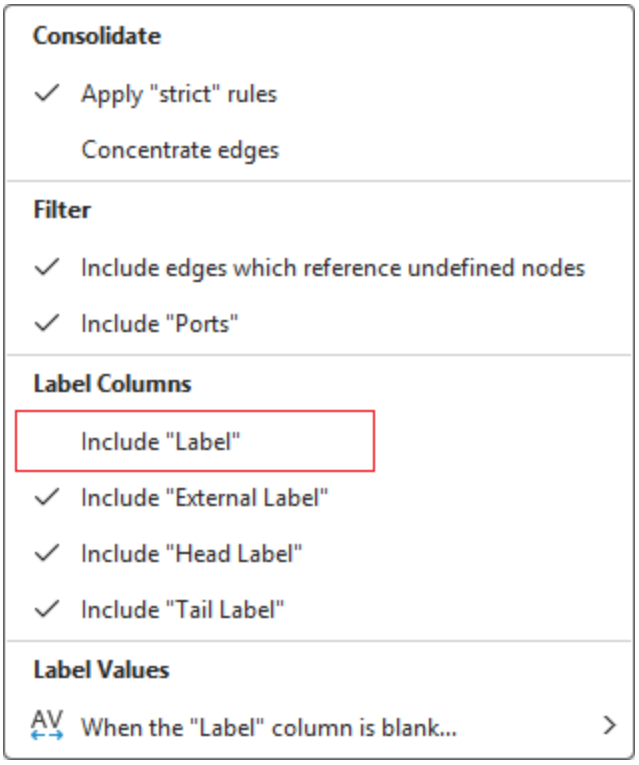
20	#	*Edge scenarios*				
21		d	cluster node to cluster node	e		
22		d	cluster node to non-cluster node	h		
23		b	cluster node to cluster border	f		lhead="cluster1"
24		c	cluster border to cluster node	e		ltail="cluster0"
25		c	cluster border to cluster border	g		ltail="cluster0" lhead="cluster1"

At this point, all the information required to generate the graph has been entered. When you press the **Refresh** button, the graph appears as follows:

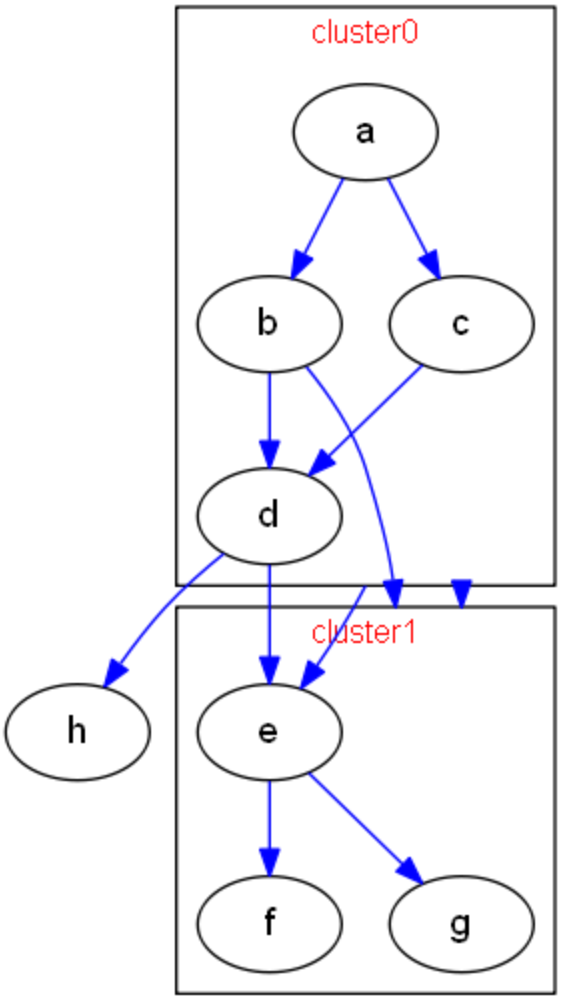


The graph now matches our intended layout. Notice that each blue edge includes a black line beneath its label, visually underlining the text. This callout-line effect comes from setting `decorate="true"` in the default `edge` keyword earlier in the spreadsheet. It is especially helpful here, since Graphviz sometimes places labels in positions that can be confusing without a visual anchor.

Now that we understand how the edge statements are rendered, we can hide the labels. The easiest way to do this is to open the **Graphviz** ribbon tab and, under **Edge Options**, clear the **Include Label** checkbox.



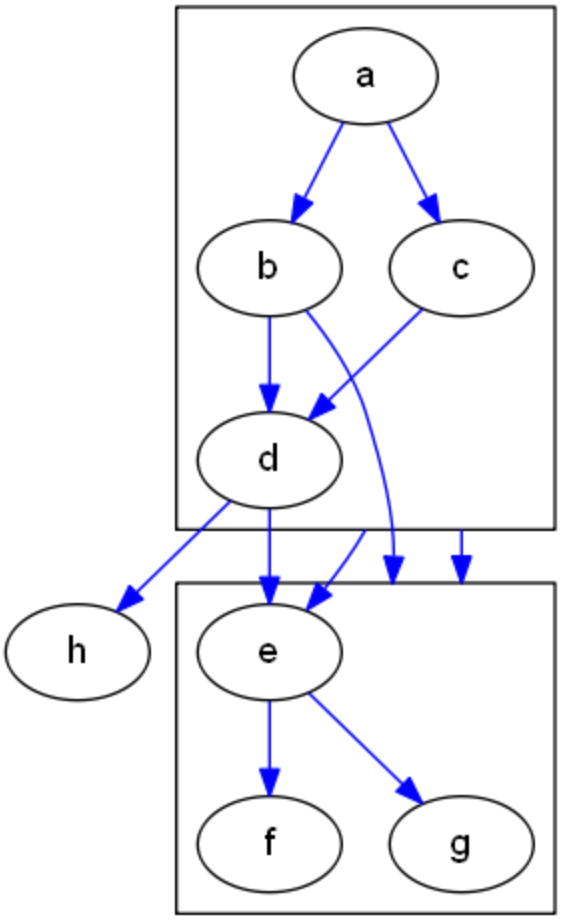
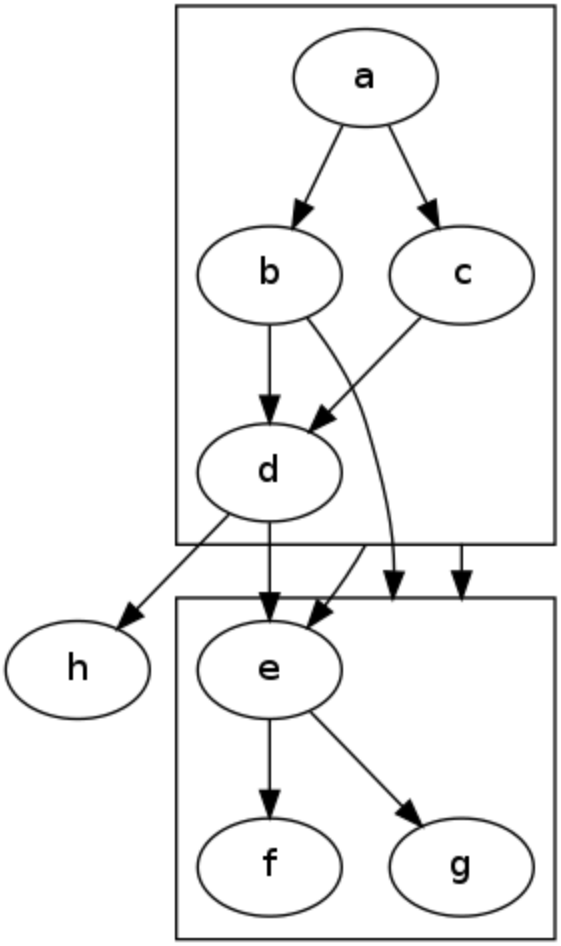
Press the `Refresh Graph` button and the graph now appears as:



This graph is now almost identical to the example from the Internet that we set out to reproduce. The final step is to remove the cluster labels that were added earlier for demonstration purposes. Since there is no settings option to toggle these labels on or off, we need to return to the *data* worksheet and delete the label entries manually so that the data appears as follows:

8	#	*cluster0*		
9		cluster0 {		
10		a		b
11		a		c
12		b		d
13		c		d
14		}		
15	#	*cluster1*		
16		cluster1 {		
17		e		g
18		e		f
19		}		

Press the `Refresh Graph` button and the graph now appears as the graph on the left; the graph we are duplicating is on the right:

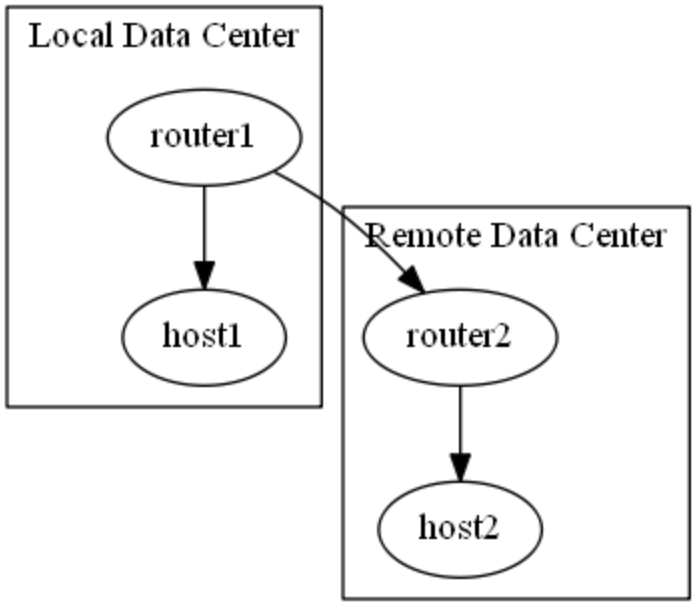
Relationship Visualizer	Stack Overflow
	

The graph was intentionally left with blue edges and a slightly smaller font size to distinguish it from the target image—different enough to show that it was not the original, yet similar enough to demonstrate that the goal was achieved.

If you want to make the final adjustments so the two graphs are truly identical, edit the style definition for the `edge` keyword on row 5 and remove the `color="blue"` attribute. Press **Refresh**, and the generated graph will match the goal image exactly.

Align Nodes Across Clusters

One of the ways Graphviz saves you time is by automatically choosing an optimal layout for nodes and edges. In many cases this produces a clean, readable diagram with no additional effort. Sometimes, however, you may want more control over placement for aesthetic or organizational reasons. Assume that you have the following graph:



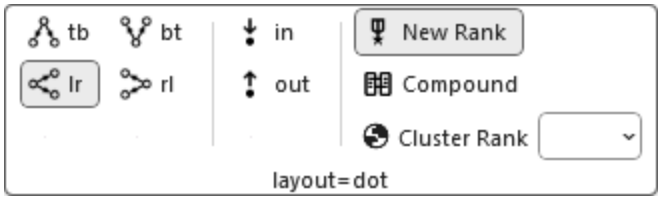
Created by this spreadsheet:

	A	B	C	G
1		**Item**	**Label**	**Is Related To Item**
2	#	*Local Data Center*		
3		{	Local Data Center	
4		router1		host1
5		}		
6	#	*Remote Data Center*		
7		{	Remote Data Center	
8		router2		host2
9		}		
10	#	*Router to Router Link*		
11		router1		router2

For aesthetic reasons, we want `router1` and `router2` to align horizontally. To accomplish this, two native Graphviz DOT commands must be added to the spreadsheet. The first appears on line 4, where we insert `newrank="true"` into the body of the main graph. This attribute—introduced in Graphviz 2.30 and not formally documented—enables an alternative ranking algorithm that allows `rank="same"` to be applied to nodes even when they belong to clusters.

	A	B	C	G
1		**Item**	**Label**	**Is Related To Item**
2	#	*newrank allows defining rank=same for nodes which belong to clusters*		
3		>	newrank="true"	

You can also achieve the same result by selection the graph option "New Rank"



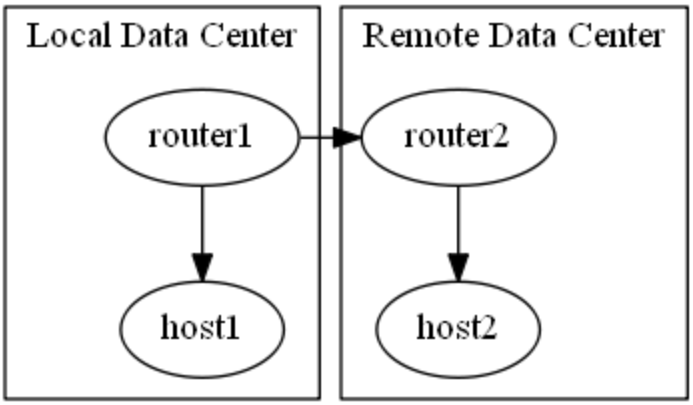
The next step is to add the following native Graphviz command after the cluster definitions:

```
{ rank="same"; "router1"; "router2"; }
```

as shown on row 13 below:

	A	B	C	G
1		**Item**	**Label**	**Is Related To Item**
2	#	*newrank allows defining rank=same for nodes which belong to clusters*		
3		>	newrank="true"	
4	#	*Local Data Center*		
5		{	Local Data Center	
6		router1		host1
7		}		
8	#	*Remote Data Center*		
9		{	Remote Data Center	
10		router2		host2
11		}		
12	#	*Align the router nodes*		
13		>	{rank="same"; "router1"; "router2";}	
14	#	*Router to Router Link*		
15		router1		router2

Press the `Refresh Graph` button, and the graph contains router1 and router2 aligned as shown below:



shape="record"

Visually, a **record** is drawn as a box whose fields are arranged as a series of horizontal or vertical sub-boxes. The **Mrecord** shape is identical to a record, except that its outer border has rounded corners. You can flip between horizontal and vertical field layouts by nesting fields inside braces `{ ... }`. By default, the top-level orientation of a record is horizontal.

For example:

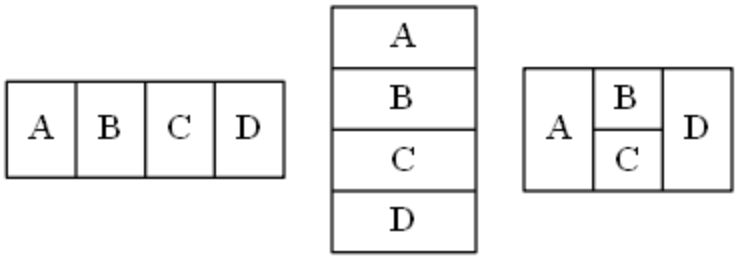
- `A | B | C | D` produces four fields arranged left to right
- `{A | B | C | D}` produces four fields arranged top to bottom
- `A | { B | C } | D` places **B** above **C**, with **A** to the left and **D** to the right

The initial orientation of a record node also depends on the value of the `rankdir` attribute.

- If `rankdir` is **Top to Bottom** or **Bottom to Top** (vertical layouts), the top-level fields in a record are displayed horizontally.
- If `rankdir` is **Left to Right** or **Right to Left** (horizontal layouts), the top-level fields are displayed vertically.

	A	B	C	G	H	I
1		Item	Label	Is Related To Item	Style Name	Extra Style Attributes
2		*node*				shape="record"
3	#	*Left to right*				
4		a	A|B|C|D			
5	#	*Top to bottom*				
6		b	{A|B|C|D}			
7	#	*Mixed*				
8		c	A|{B|C}|D			

Results in:



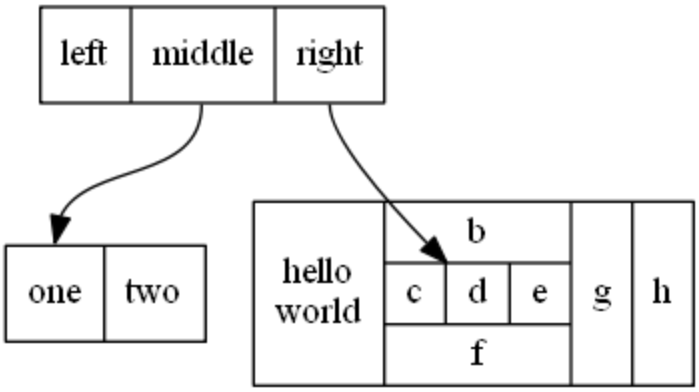
`shape="record"` with Ports

You can specify port identifiers as part of the field values in a record shape. The first string in the field ID assigns a port name to that field, which can then be combined with the node name to control where an edge attaches. The second string provides the visible text for the field and supports the usual escape sequences `\n`, `\l`, and `\r`.

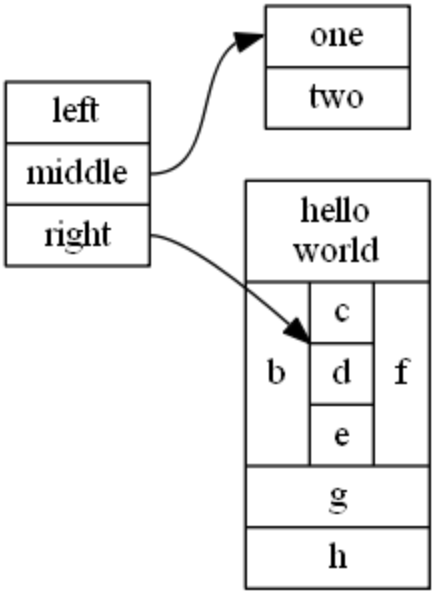
Therefore, if we specify the following (with color-coding added here to highlight the ports):

	A	B	C	G	H	I
1		**Item**	**Label**	**Is Related To Item**	**Style Name**	**Extra Style Attributes**
2		***node***				shape="record"
3						
4		struct1	<f0> left | <f1> middle | <f2> right			
5		struct2	<f0>one | <f1>two			
6		struct3	hello world | { b | { c | <here> d | e } | f } | g | h			
7		struct1:f1		struct2:f0		
8		struct1:f2		struct3:here		

We will generate the following graph:



If we change the graphing direction from **Top to Bottom** to **Left to Right** on the *Settings* worksheet and regenerate the graph, it will appear as follows:



Consolidating Edges

Some sets of data will naturally produce multiple edges between the same pair of nodes. A good example is the U.S. state border relationships. Every pair of neighboring states has two relationships: for instance, Michigan borders Ohio, and Ohio borders Michigan.

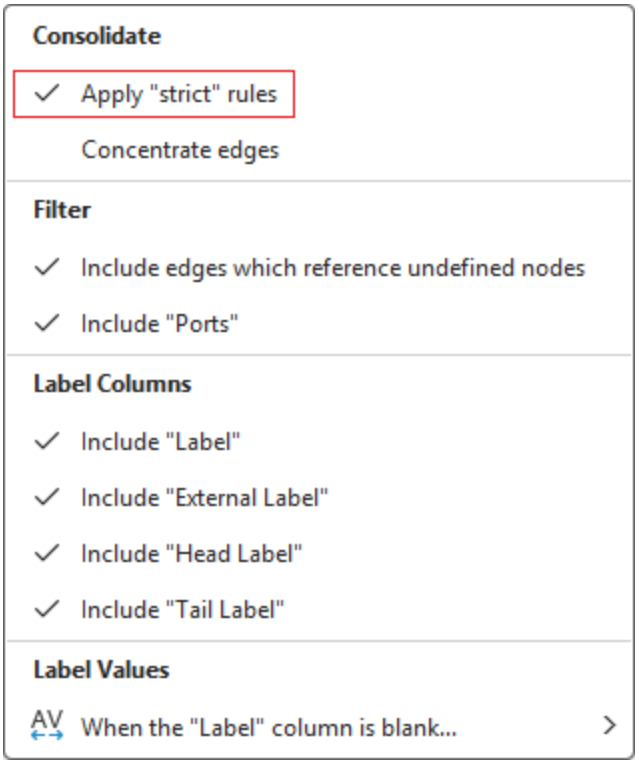
When these relationships are plotted as an undirected graph, the following data:

	A	B	D	H
1		**Item**	**Label**	**Related Item**
2				
3		Michigan		Ohio
4		Michigan		Ohio

Generates the following graph:

Graphviz can consolidate these duplicate relationships into a single edge. When Graphviz is instructed that the graph is `strict`, multiple edges between the same pair of nodes are not permitted.

To enable this behavior, open the **Graphviz** ribbon tab and, in the **Edge Options** section, set **Apply "strict" rules** to **Yes**, as shown below:



Press the `Refresh Graph` button and the graph appears as:

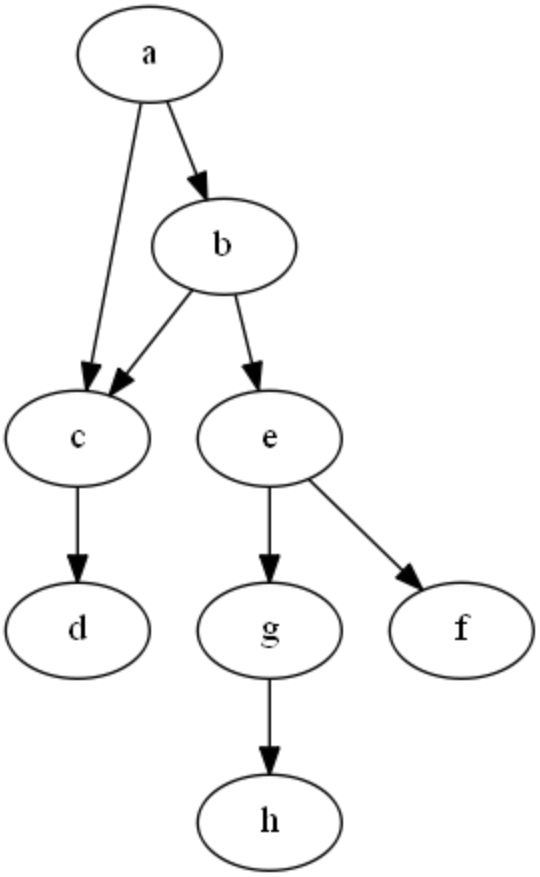
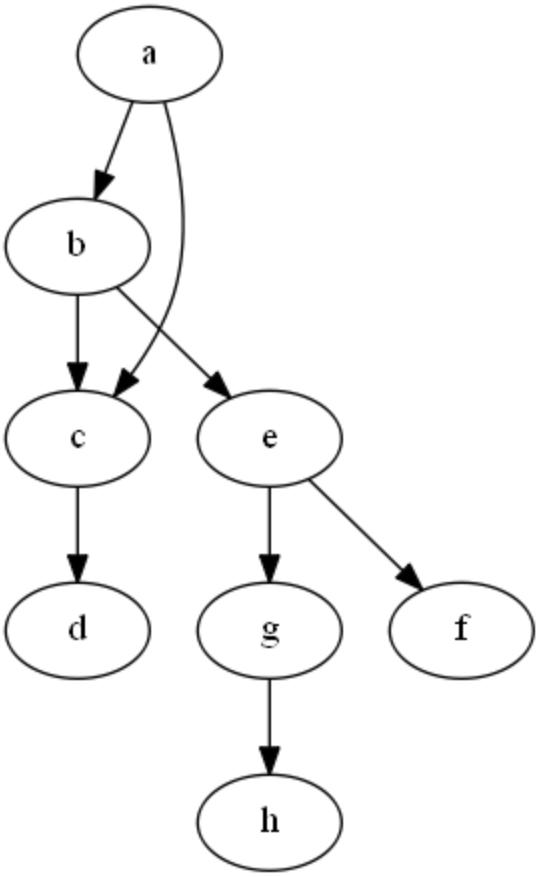
Changing the Order of Edges

Differences in style can be achieved by altering the edge ordering. If the value of the `ordering` attribute is `"out"`, then the outgoing edges of a node—that is, edges for which the node is the tail—must appear left-to-right in the same order in which they are defined in the input. If the value is `"in"`, then the incoming edges of a node must appear left-to-right in the order in which they are defined.

Assume you have several edge relationships defined as follows:

	A	B	C	G	H	I
1		Item	Label	Is Related To Item	Style Name	Extra Style Attributes
2		*graph*				ordering="in"
3		a		b		
4		a		c		
5		b		c		
6		c		d		
7		b		e		
8		e		g		
9		e		f		
10		g		h		

Pressing the `Refresh Graph` button, the graph appears as:

`ordering="in"`	`ordering="out"`
	

Edge Head and Tail Labels

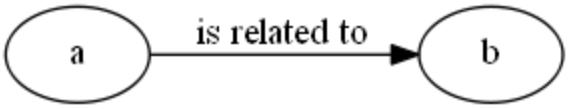
The default view of the Relationship Visualizer provides a **Label** column for assigning a standard edge label. Graphviz also supports placing labels at the tail and/or head of an

edge using the `taillabel` and `headlabel` attributes, respectively. The *data* worksheet includes dedicated columns for these attributes, but they are hidden by default. To use them, simply unhide the corresponding columns.

For example, if we have a simple relationship such as:

	A	B	D	G
1		**Item**	**Label**	**Related Item**
2				
3		a	is related to	b

Producing the graph:



We can click the `Show/Hide Columns` dropdown list on the **Graphviz** tab to expose the additional label columns

→

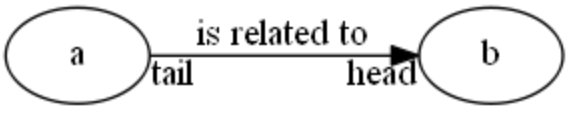
The data worksheet now appears as:

	A	B	C	D	E	F	G
1		**Item**	**Tail Label**	**Label**	**Outside Label**	**Head Label**	**Related Item**
2							
3		a		is related to			b

If we place the value `tail` at the tail of an edge, and `head` at the head of an edge, the data in the spreadsheet would look as follows:

	A	B	C	D	E	F	G
1		**Item**	**Tail Label**	**Label**	**Outside Label**	**Head Label**	**Related Item**
2							
3		a	tail	is related to		head	b

Pressing the `Refresh` button creates the following graph:



Edges to the Node Center

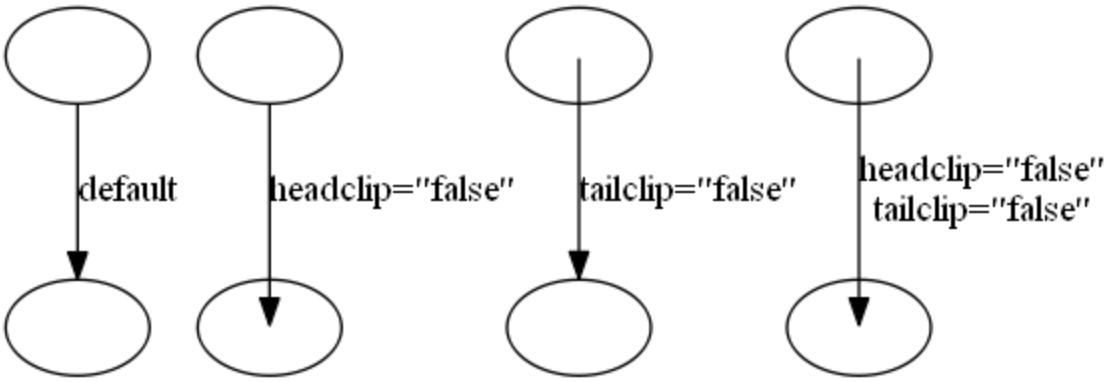
By default, an edge is clipped to the boundary of a node's shape. You can override this behavior so that an edge begins and/or ends at the center of the node instead of its perimeter. The `headclip` and `tailclip` attributes control this behavior. When set to `true`

(the default), the edge is clipped to the node boundary. When set to `false`, the edge connects to the center of the node—or to the center of a port, if one is specified.

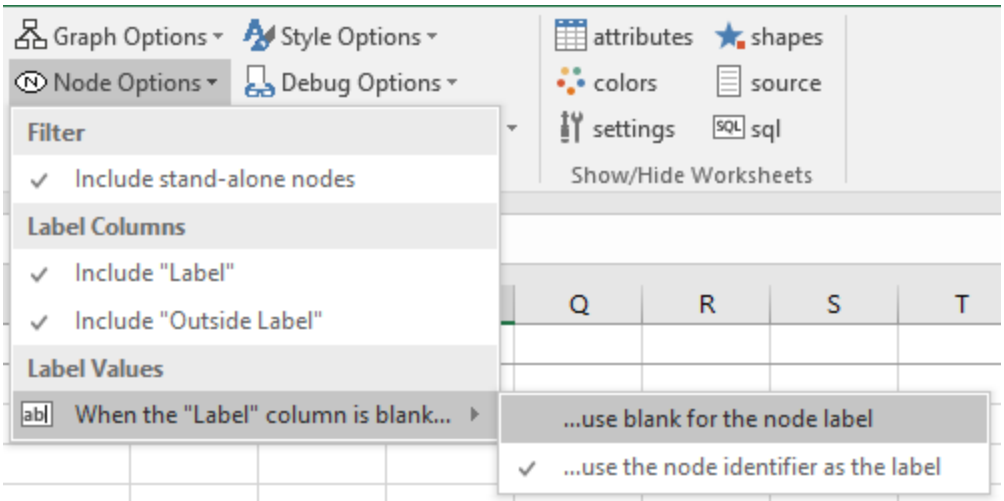
The data below:

	A	B	C	G	H	I
1		Item	Label	Is Related To Item	Style Name	Extra Style Attributes
2		a,b,c,d,e,f,g,h				
3		a	default	b		
4		c	headclip="false"	d		headclip="false"
5		e	tailclip="false"	f		tailclip="false"
6		g	headclip="false" tailclip="false"	h		headclip="false" tailclip="false"

Creates the following graph:



Note: *The nodes have blank labels to make the illustration of edges coming from or going to the center of the node easier to see. Enabling the 'Nodes' graph option 'When the* `Label` *column is blank...' '...use blank for the node label" on the* `Graphviz` *ribbon tab is required to achieve this effect.*

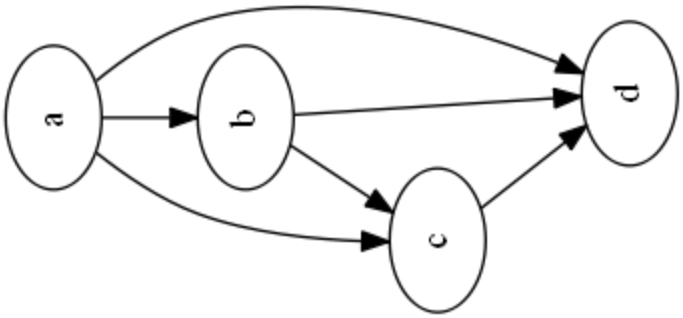


Rotating Graphs 90 Degrees

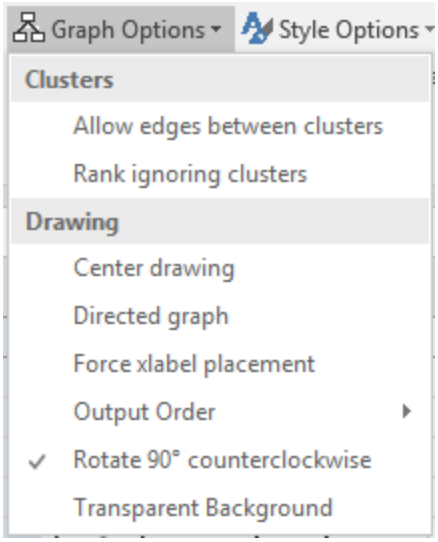
Graphs can have their drawing orientation set to landscape by setting the rotate attribute equal to 90. The final output is rotated in the counterclockwise direction. The data below:

	A	B	C	G	H	I
1		Item	Label	Is Related To Item	Style Name	Extra Style Attributes
2		*graph*				rotate="90"
3		a		b		
4		a		c		
5		a		d		
6		b		c		
7		b		d		
8		c		d		

Creates the following graph:



An alternate method is to check the "Rotate 90 counterclockwise" option from the Graph Options on the Graphviz tab.



SQL Tools

Relationship Visualizer can query data from Excel spreadsheets and Access databases using standard SQL. You can pull data from either source, or combine both, before generating Graphviz diagrams.

This feature is *optional* and available only on Windows.[1] It lets you write SQL queries that span multiple worksheets or tables and produce graph-ready results.

Use SQL when you want to:

- Combine data from several Excel worksheets
- Query Access tables directly
- Mix Excel and Access data within a workflow
- Automate repetitive data-preparation tasks

SQL Topics

Use the links below to jump directly to key SQL concepts and features used throughout the Relationship Visualizer.

SQL Extensions

These SQL extensions simplify common graph-building tasks. Passing specific values as SQL parameters activates each utility feature.

- Directives
- Group Items into Nested Clusters
- Split Long Label Text
- Generate Edges to Chain Nodes
- Align Nodes on the Same Level
- Traverse Trees Recursively
- Iterate SQL Results
- Enumerate Values
- Count Substitution
- Concatenation

SQL Advanced Examples

These SQL examples illustrate how to build graphs using SQL statements and extensions.

- Create Organization Charts
- Build Roadmaps and Timelines

[1]: The SQL feature is **Windows only** and requires basic SQL knowledge. If you use macOS, you can still use Relationship Visualizer's manual data entry options.

What is SQL?

Structured Query Language (SQL) is a domain-specific language designed for stream processing in a Relational Data Management System (RDMS). It is particularly useful in handling structured data where there are relations between different entities of the data.

SQL `SELECT` statements are designed to query data contained in a relational database. The SQL `SELECT` statement fetches data from a database, by specifying column headings, and tables from which records are to be selected, and the criteria for selection.

`SELECT *` is used to select all columns from a table. The `FROM` clause specifies the Table from which columns are to be selected. The optional `WHERE` clause specifies the criteria for selection.

Syntax:

```sql
SELECT column_name FROM table_name [WHERE clause]
```

Excel SQL Queries

Excel can act as a flat-file RDBMS, accessible through ActiveX Data Objects (ADO) and SQL queries. Its SQL support is provided through a dedicated Excel-specific dialect implemented by the Jet and ACE database engines.

> **TIP**
>
> See SQL Syntax for detailed information on the Excel SQL dialect.

The Relationship Visualizer includes a worksheet named `sql` that abstracts away most of the complexity involved in writing ADO SQL. It lets you write SQL `SELECT` statements using ADO-compatible syntax, while the tool automatically manages all connection details. Queries can reference either the Relationship Visualizer workbook itself or any other Excel workbook as the data source.

When a query is executed, the results are written to the `data` worksheet, where they become immediately available for graphing.

ADO supports many capabilities, but the Relationship Visualizer uses only a focused subset —specifically SQL `SELECT` queries of the form:

```sql
SELECT [Country Code] AS [Item],
       [Country Name] AS [Label]
FROM   [Countries$]
```

This is a standard SQL query—one that selects two columns from the database (which, in this context, is an Excel worksheet) and assigns names to the results that match the column names on the `data` worksheet. As with a traditional database, the query specifies the name of a table; however, Excel introduces two small differences. The worksheet name must be enclosed in square brackets, and the actual sheet name—**Countries** in this example—must have a **$** appended to it.

Square brackets are also required around column names, since Excel headers may contain spaces or special characters. Bracketing ensures ADO interprets the text as a column identifier. Quoted values cannot be used for this purpose, because quotes indicate string literals rather than column references.

Map results to `data` worksheet columns

After the `SELECT` statement runs, the Relationship Visualizer processes the results and places the selected values into the corresponding columns of the `data` worksheet. Only recognized column names are used; all other fields are ignored. The default English column names on the `data` worksheet are:

- `[Comment]`
- `[Item]`
- `[Label]`
- `[External Label]`
- `[Tail Label]`
- `[Head Label]`
- `[Related Item]`

- `[Style Name]`
- `[Attributes]`

String constants

Excel SQL also allows you to specify string values as selection columns. When used in a `SELECT` statement, the string is repeated for every row returned by the query. This is particularly useful for assigning constant values **such as a style name** to each row of the result set.

For example, suppose the `styles` worksheet defines a style named `Country`. You can modify the query to include this style by adding a constant string column, ensuring that every row in the output is tagged with the `Country` style:

```sql
SELECT [Country Code] as [Item],
       [Country Name] as [Label],
       'Country'      as [Style Name]
FROM   [Countries$]
```

Apply search conditions

SQL allows you to specify `WHERE` conditions to control which records are returned. The `WHERE` clause defines the qualifying criteria for a query. Multiple conditions can be combined using the `AND` and `OR` operators, with optional parentheses to group expressions. Only rows that meet the specified conditions are included in the results.

For example, suppose we want a list of countries that use US Dollars as their national currency. The international code for US Dollars is `USD`, so the SQL query would be modified as follows:

```sql
SELECT [Country Code] as [Item],
       [Country Name] as [Label],
       'Country' as [Style Name]
FROM   [Countries$]
WHERE  [Currency Code] = 'USD'
```

Eliminate duplicate values

A column often contains duplicate values. To list only the unique entries, use the `SELECT DISTINCT` clause. The `DISTINCT` keyword filters the result set so that each value appears only once, even if it occurs many times in the underlying data.

For example, if the worksheet includes a column listing the continent for each country, you can retrieve a deduplicated list of continents using a query such as:

```sql
SELECT DISTINCT [Continent] as [Item],
       [Continent]          as [Label],
       'Continent'          as [Style Name]
FROM   [Countries$]
```

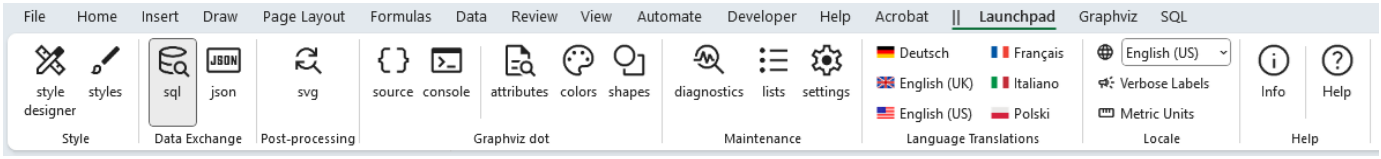
How to Use SQL in the Relationship Visualizer

Follow these simple steps to create graphs using SQL queries in Relationship Visualizer:

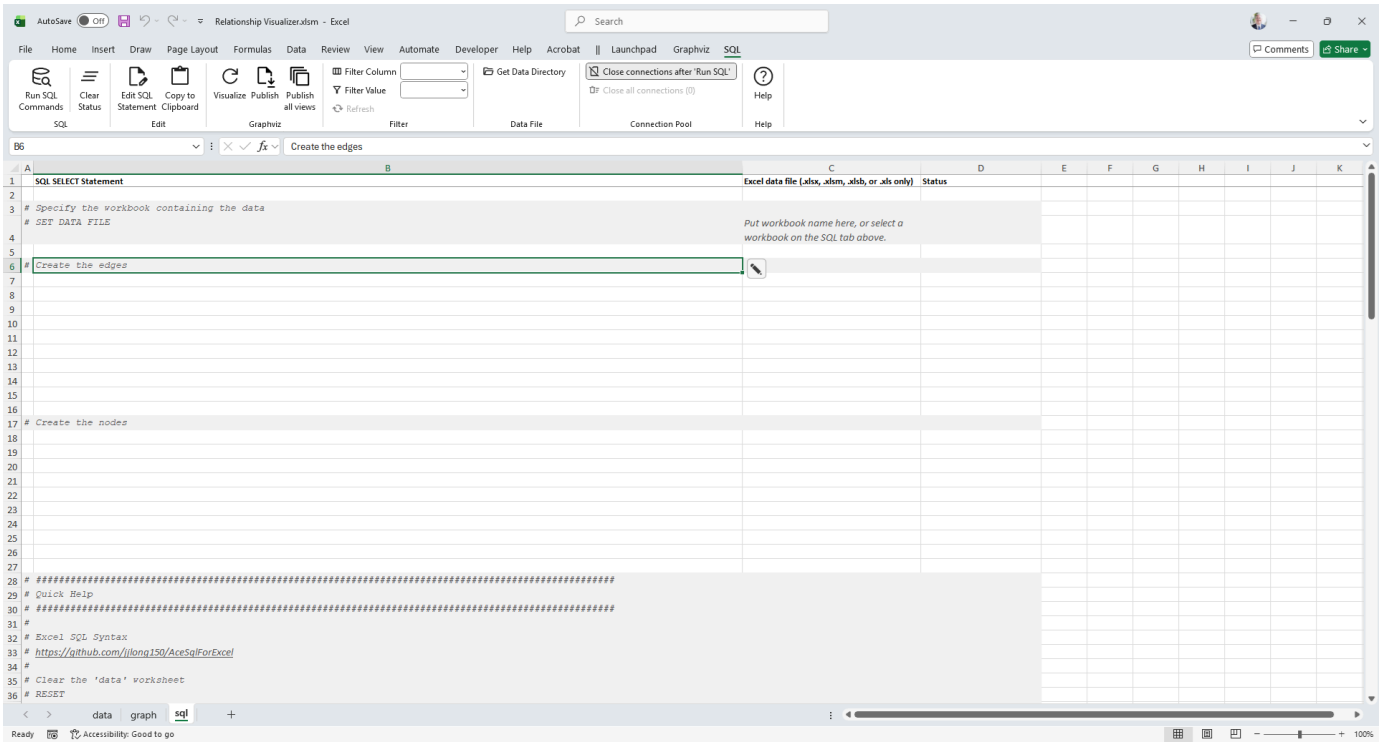
Open the `sql` Worksheet

In your Relationship Visualizer workbook, use the Launchpad to show the `sql` worksheet, then make the `sql` worksheet active.

`Launchpad`



`SQL Worksheet`



Write Your SQL Query

Enter a SQL `SELECT` query using ADO syntax. For example, to create nodes for countries in the world, a simple query would look as follows:

```sql
SELECT [Country Code] as [Item],
       [Country Name] as [Label]
FROM   [Countries$]
```

Reminders:

- Use square brackets `[ ]` for column and worksheet names.
- Worksheet names end with `$` (e.g., `[Countries$]`).

Assign column names from the data workbook to the Relationship Visualizer column names on the `data` worksheet. The default column names are:

- `[Comment]`
- `[Item]`
- `[Label]`
- `[External Label]`
- `[Tail Label]`
- `[Head Label]`
- `[Related Item]`
- `[Style Name]`
- `[Attributes]`

Specify the Data File

Data can be queried from any Excel workbook or Access database. Specify the datasource filename in column C.

If you leave column C blank the queries will try to use a workbook specified in the Ribbon. If not present there either, it default to looking for the worksheets in the Relationship Visualizer workbook.

Run the Query

Click the `Run SQL Commands` button to execute your query. The results will populate the `data` worksheet, with status messages in column D (e.g., **SUCCESS**).

Visualize the Graph

Press the `Refresh` button to generate your Graphviz visualization based on the query results.

The `SQL` Worksheet

The `sql` worksheet is the worksheet you will write and execute queries to pull data from another Excel workbook.

Before we run any SQL, lets gain an understanding of the mandatory and optional columns on this worksheet.

The `data` Worksheet has 4 columns (A-D):

A	B	C	D
Indicator	SQL SELECT Statement	Excel data file	Status

Indicator

The `Indicator` column is used to draw special attention to a row.

- A `#` hash character treats the row as a comment. The text in the row will turn gray, and this row will be skipped when running the SQL.
- An `!` exclamation mark character will appear if errors are detected in your data on this row. The Status will turn red, and an error message will be displayed in the Status column.

SQL SELECT Statement

This column is where the SQL Statement is entered. For example:

```sql
SELECT [Country Code] as [Item],
       [Country Name] as [Label]
FROM   [Countries$]
```

Excel data file

This column specifies the Excel file that contains the data.

- If the column is left blank, the current Relationship Visualizer workbook is used as the data source.
- If a fully qualified filename (path + file) is provided, that file is used exactly as specified.
- If a relative filename (file only) is provided, it is assumed to be located in the same directory as the Relationship Visualizer workbook.

Status

The result of the SQL query. The value will be either:

- 🟩 `SUCCESS` - The query was executed successfully.
- 🟥 `FAILURE` - The query failed. Any error message will be appended to the status value.

  Even simple Excel SQL queries can fail for reasons that aren't obvious at first glance. Keep an eye out for these common issues:

  - **Missing the `$` in worksheet names**
    ADO requires worksheet names to end with a `$` (e.g., `[Countries$]`). Omitting it will cause the query to fail.
  - **'xyz' is not a valid name. Make sure that it does not include invalid characters or punctuation and that it is not too long** *(where 'xyz' is a worksheet name in your `FROM` clause)*.

    This error usually means you have referenced a worksheet that cannot be found for one of these reasons:

    - **The worksheet name is misspelled** Verify that the name in your FROM clause exactly matches the worksheet tab name, including capitalization and spacing.
    - **The worksheet does not exist in the Relationship Visualizer workbook.** Ensure that the worksheet is present and has not been renamed or deleted.
    - **No external Excel workbook was specified as a data source.** ADO cannot resolve worksheet names unless a workbook has been specified.

    - **The wrong Excel workbook was selected as the data source.** Double-check that the workbook you chose contains the worksheet referenced in your query.

- **Forgetting square brackets around sheet or column names**
  Brackets are required when names contain spaces, punctuation, or non-alphanumeric characters.
  Examples: `[Country Code]`, `[Sales 2024$]`.

- **Using quotes instead of brackets for identifiers**
  `"Country Code"` is treated as a string literal, not a column reference. Always use `[Country Code]`.

- **Header row not in row 1**
  ADO expects column headers in the first row of the worksheet. If your data starts lower, the query may return no rows or incorrect results.

- **Mixed data types in a column**
  Excel may infer the wrong type (e.g., treating numeric IDs as text). This can cause filtering or comparison operations to behave unexpectedly.

- **Sheet names that look like ranges**
  Names such as `A1` or `R1C1` can confuse the provider. Renaming the sheet usually resolves the issue.

- **Hidden or filtered rows**
  ADO reads the entire sheet, including hidden rows. If you expect filtered results, you must filter them in SQL.

- **External workbook paths not fully pathed**
  When referencing another workbook, the full path must be provided.

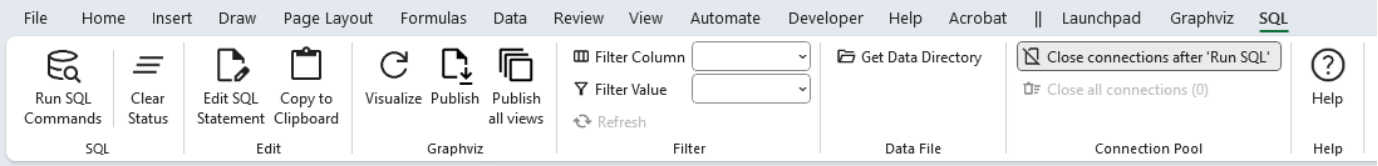
- ☐ `SKIPPED` - The row was not processed because either a comment indicator (`#`) was present in Column A, or a Filter was applied and the row did not meet the filter criteria. Rows can be skipped for two independent reasons:

    - Column A begins with `#`, the row is treated as a comment and is never processed.

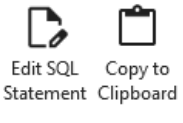
- A Filter is active, and only rows that match the filter criteria are executed; all others are skipped.

The `SQL` Ribbon Tab

The `SQL` ribbon tab is activated whenever the `sql` worksheet is activated. It appears as follows:

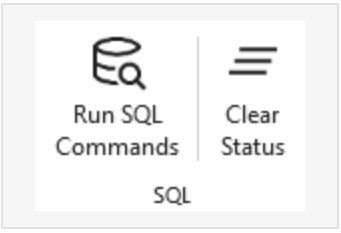


It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
SQL		The **SQL** controls let you execute the queries defined on this worksheet and manage the status information associated with each statement.
Edit		Provides tools to help get around Excel's inability to display large amounts of cell text.
Graphviz		The `sql` worksheet includes several convenience buttons designed to streamline graph generation after running SQL queries.
Filter		The **Filter** controls allow you to limit which rows are processed when running SQL statements.

Group	Controls	Description
Data File		The **Data File** controls let you point your SQL queries at external Excel workbooks.
Connection Pool		The Connection Pool group manages how ADO workbook connections are reused, opened, and closed during SQL batch execution to improve performance and control file access.
Help		Provides a link to the `Help` content for the `sql` worksheet (i.e. this web page).

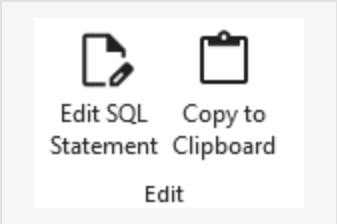
SQL



The **SQL** controls let you execute the queries defined on this worksheet and manage the status information associated with each statement.

Label	Control Type	Description
Run SQL Commands	Button	Sequentially executes all SQL statements in the worksheet and writes the results to the `data` worksheet.
Clear Status	Button	Clears all values in the **Status** column, allowing you to reset or re-run SQL statements cleanly.

Edit



Provides tools to help get around Excel's inability to display large amounts of cell text.

Label	Control Type	Description
Edit SQL Statement	Button	Launches the Edit Text form with the contents of the currently selected cell. A second location where the **Edit SQL Statement** button appears is as a floating pencil button on the right side of any selected SQL cell.  Clicking the pencil button performs the same action as selecting the **Edit SQL Statement** button in the Ribbon.
Copy to Clipboard	Button	Copies the contents of the cell as straight text to the Microsoft Windows clipboard, so it can be pasted into an external editor. Characters such as quotes are not escaped as would occur when using Excel's copy (Ctrl+C).

Graphviz

Convenience buttons designed to streamline graph generation after running SQL queries. These controls mirror the commands on the Graphviz ribbon tab but place them closer to where SQL work is performed:

Label	Control Type	Description
Visualize	Button	Graphs the data in the `data` worksheet and displays the results in Excel. This button mirrors the **Visualize** command on the Graphviz ribbon tab and is provided here for faster graph creation after running SQL queries.
Publish	Button	Graphs the data in the `data` worksheet and writes the output to a file using the settings on the Graphviz ribbon tab. This duplicates the **Publish** command and streamlines exporting graphs after executing SQL queries.
Publish all views	Button	Graphs the data in the `data` worksheet using all defined views and writes each graph to a file based on the Graphviz ribbon settings. This matches the Graphviz ribbon's **Publish all views** command and accelerates batch graph generation.

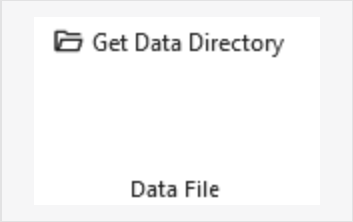
Filter

The **Filter** controls allow you to limit which rows are processed when running SQL statements. This is especially useful when working with large datasets or when you want to preview results for a specific subset of data.

Label	Control Type	Description
Filter Column	Dropdown list	Lists column letters **E–Z**. Select the column you want to filter on.
Filter Value	Dropdown list	Displays all unique (de-duplicated) values found in the selected **Filter Column**. When a value is chosen, SQL statements will run **only** on rows where that column matches the selected value.
Refresh	Button	Updates the **Filter Value** list. Use this after changing data in the selected Filter Column to ensure the dropdown reflects the latest values.

Data File

The **Data File** controls let you point your SQL queries at external Excel workbooks. This is especially useful when your data is stored in multiple files—such as monthly extracts, departmental exports, or versioned snapshots—and you want to run the same SQL queries against each file without rewriting anything.

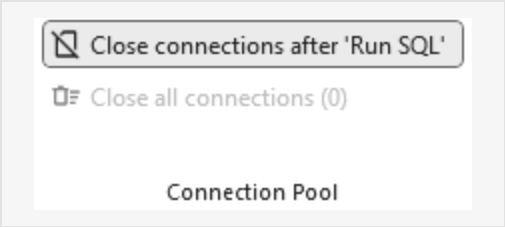


Label	Control Type	Description
Get Data Directory	Button	Opens a directory-picker dialog. When a folder is selected, its path replaces the “Get Data Directory” label.
File Name	Dropdown list	Lists the Excel files found in the selected directory that are suitable for running Excel SQL queries against. This list is only visible when a data directory has been specified. **NOTE:** Entries in the file name column, and `SET DATA FILE`

Label	Control Type	Description
		statements take precedence when resolving conflicts over which data source to use.
Reset	Button	Clears the data directory and file name selections. This button is only visible when selections have been made.

Why would I use this?
Many workflows produce recurring Excel extracts—for example, one workbook per month, quarter, or reporting cycle. By selecting a directory and choosing a file from the dropdown, you can run the *same* SQL queries against each extract in turn. This makes it easy to compare periods, regenerate graphs for new data, or batch-process multiple datasets without modifying your SQL.

Connection Pool



The **Connection Pool** group manages how ADO workbook connections are reused, opened, and closed during SQL batch execution to improve performance and control file access.

Connection pooling was added in Version 7.0 in response to a March 2025 Office update that caused ADO connections to take over 12 seconds (previously under 4 milliseconds). See: Excel ADO connection issue in recent Office 365 update ↗

Workbook connections are now reused across all SQL statements during a `Run SQL Statements` batch run.

Users can choose to close connections after batch execution or keep them open until manually closed or until the workbook exits.

Note: Keeping connections open may improve performance but can prevent access to referenced workbooks.

Label	Control Type	Description
Close connections after 'Run SQL'	Toggle Button	Determines whether connections remain open (pooled) after running SQL statements or are closed immediately after execution.
Close all connections (#)	Button	Displays the number of currently open connections. Enabled only when one or more connections are open. Pressing the button closes all open connections.

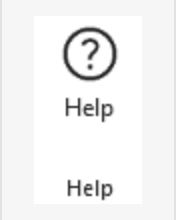
When should I keep connections open?

Keeping connections open (pooled) is most beneficial when running many SQL statements in a batch, especially if each statement targets external workbooks. Reusing the same connection avoids repeated ADO initialization overhead and can dramatically improve performance.

However, if you need to open, edit, or replace any referenced workbooks, you may prefer to close connections after each run to avoid file-locking issues.

Help

Provides a link to the `Help` content for the `sql` worksheet (i.e. this web page).

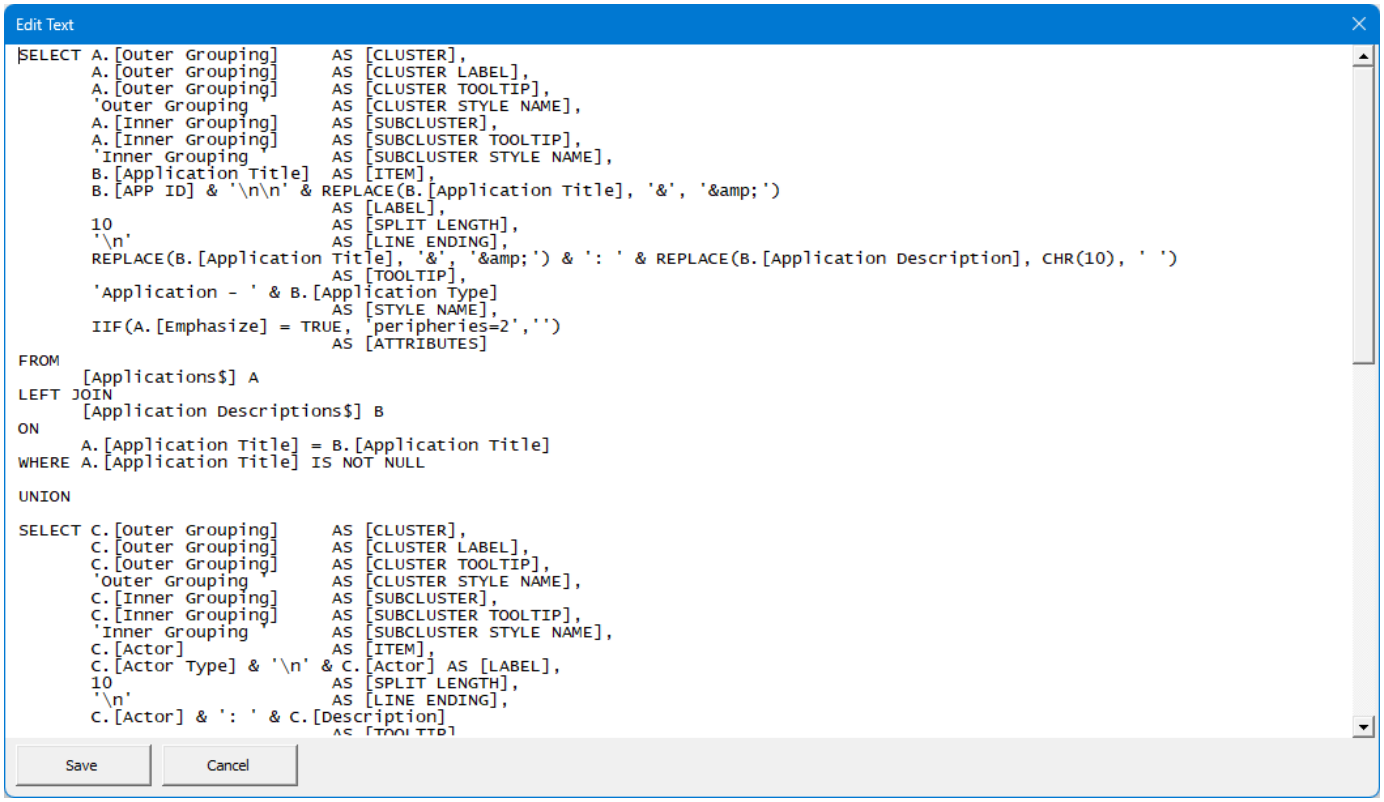


Label	Control Type	Description
Help	Button	Provides a link to this web page.

The `Edit SQL Statement` Form

Excel rows can hold a great deal of information, but cells have their quirks. Each cell can store up to 32,767 characters, yet only about 1,024 of them are visible directly in the grid without entering cell-edit mode. For rows, the practical display limit depends on the combined content of each cell, though in reality screen size and readability become limiting factors long before Excel's technical limits are reached. These limits get reached quickly if you like to break SQL statements down over multiple lines. In short, you can store a lot—but you may not be able to see it all at once.

To address this, Version 7 introduced an `Edit SQL Statement` form. When you press the **Edit SQL Statement** button, a modal window opens containing the full text of the currently selected cell, making it easier to view and edit long content without fighting Excel's display constraints.



Horizontal and vertical scroll bars are provided to help navigate the text. You can change the text within the form.

Pressing the `Save` button transfers the contents from the form back to the active cell.

SQL to Graph Example

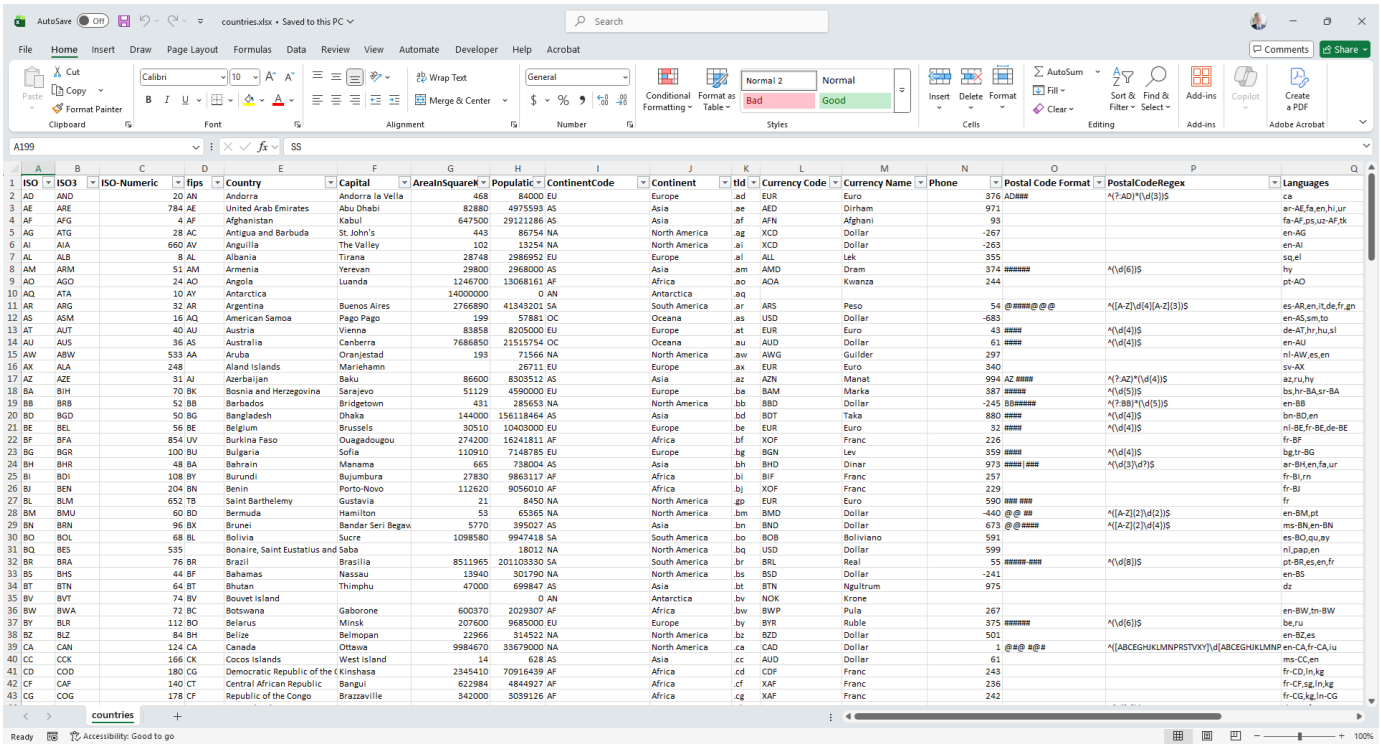
The easiest way to learn SQL-driven graph generation is by example. The `15 - Using SQL - World Map` sample in the `samples` folder will guide our walkthrough of the `sql` worksheet.

Our goal is to create a logical world map by graphing each country, linking it to its continent, and assigning a distinct color to every continent.

The `SQL - World Map` directory contains:

- A copy of the Relationship Visualizer workbook with SQL statements already defined
- Preconfigured styles on the `styles` worksheet
- A `countries.xlsx` workbook containing the data we will query

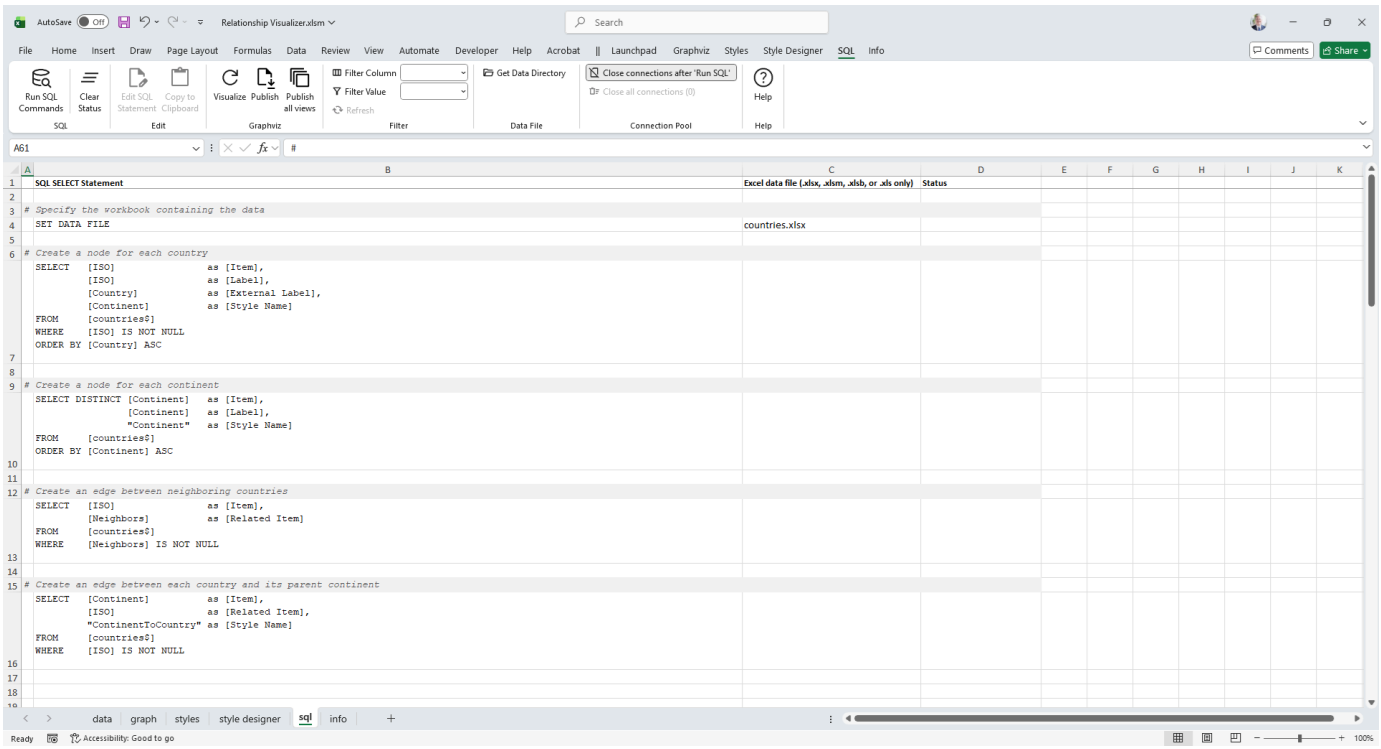
An example of the `countries` workbook appears as follows:



ISO	ISO3	ISO-Numeric	fips	Country	Capital	AreaInSquareK	Populatic	ContinentCode	Continent	tld	Currency Code	Currency Name	Phone	Postal Code Format	PostalCodeRegex	Languages
AD	AND	20	AN	Andorra	Andorra la Vella	468	84000	EU	Europe	.ad	EUR	Euro	376	AD###	^(?:AD)*(\d{3})$	ca
AE	ARE	784	AE	United Arab Emirates	Abu Dhabi	82880	4975593	AS	Asia	.ae	AED	Dirham	971			ar-AE,fa,en,hi,ur
AF	AFG	4	AF	Afghanistan	Kabul	647500	29121286	AS	Asia	.af	AFN	Afghani	93			fa-AF,ps,uz-AF,tk
AG	ATG	28	AC	Antigua and Barbuda	St. John's	443	86754	NA	North America	.ag	XCD	Dollar	-267			en-AG
AI	AIA	660	AV	Anguilla	The Valley	102	13254	NA	North America	.ai	XCD	Dollar	-263			en-AI
AL	ALB	8	AL	Albania	Tirana	28748	2986952	EU	Europe	.al	ALL	Lek	355			sq,el
AM	ARM	51	AM	Armenia	Yerevan	29800	2968000	AS	Asia	.am	AMD	Dram	374	######	^(\d{6})$	hy
AO	AGO	24	AO	Angola	Luanda	1246700	13068161	AF	Africa	.ao	AOA	Kwanza	244			pt-AO
AQ	ATA	10	AY	Antarctica		14000000	0	AN	Antarctica	.aq						
AR	ARG	32	AR	Argentina	Buenos Aires	2766890	41343201	SA	South America	.ar	ARS	Peso	54	@####@@@	^([A-Z]\d{4}[A-Z]{3})$	es-AR,en,it,de,fr,gn
AS	ASM	16	AQ	American Samoa	Pago Pago	199	57881	OC	Oceana	.as	USD	Dollar	-683			en-AS,sm,to
AT	AUT	40	AU	Austria	Vienna	83858	8205000	EU	Europe	.at	EUR	Euro	43	####	^(\d{4})$	de-AT,hr,hu,sl
AU	AUS	36	AS	Australia	Canberra	7686850	21515754	OC	Oceana	.au	AUD	Dollar	61	####	^(\d{4})$	en-AU
AW	ABW	533	AA	Aruba	Oranjestad	193	71566	NA	North America	.aw	AWG	Guilder	297			nl-AW,es,en
AX	ALA	248		Aland Islands	Mariehamn		26711	EU	Europe	.ax	EUR	Euro	340			sv-AX
AZ	AZE	31	AJ	Azerbaijan	Baku	86600	8303512	AS	Asia	.az	AZN	Manat	994	AZ ####	^(?:AZ)*(\d{4})$	az,ru,hy
BA	BIH	70	BK	Bosnia and Herzegovina	Sarajevo	51129	4590000	EU	Europe	.ba	BAM	Marka	387	#####	^(\d{5})$	bs,hr-BA,sr-BA
BB	BRB	52	BB	Barbados	Bridgetown	431	285653	NA	North America	.bb	BBD	Dollar	-245	BB######	^(?:BB)*(\d{5})$	en-BB
BD	BGD	50	BG	Bangladesh	Dhaka	144000	156118464	AS	Asia	.bd	BDT	Taka	880	####	^(\d{4})$	bn-BD,en
BE	BEL	56	BE	Belgium	Brussels	30510	10403000	EU	Europe	.be	EUR	Euro	32	####	^(\d{4})$	nl-BE,fr-BE,de-BE
BF	BFA	854	UV	Burkina Faso	Ouagadougou	274200	16241811	AF	Africa	.bf	XOF	Franc	226			fr-BF
BG	BGR	100	BU	Bulgaria	Sofia	110910	7148785	EU	Europe	.bg	BGN	Lev	359	####	^(\d{4})$	bg,tr-BG
BH	BHR	48	BA	Bahrain	Manama	665	738004	AS	Asia	.bh	BHD	Dinar	973	####|###	^(\d{3}\d?)$	ar-BH,en,fa,ur
BI	BDI	108	BY	Burundi	Bujumbura	27830	9863117	AF	Africa	.bi	BIF	Franc	257			fr-BI,rn
BJ	BEN	204	BN	Benin	Porto-Novo	112620	9056010	AF	Africa	.bj	XOF	Franc	229			fr-BJ
BL	BLM	652	TB	Saint Barthelemy	Gustavia	21	8450	NA	North America	.gp	EUR	Euro	590	### ###		fr
BM	BMU	60	BD	Bermuda	Hamilton	53	65365	NA	North America	.bm	BMD	Dollar	-440	@@ ##	^([A-Z]{2}\d{2})$	en-BM,pt
BN	BRN	96	BX	Brunei	Bandar Seri Begaw	5770	395027	AS	Asia	.bn	BND	Dollar	673	@@#####	^([A-Z]{2}\d{4})$	ms-BN,en-BN
BO	BOL	68	BL	Bolivia	Sucre	1098580	9947418	SA	South America	.bo	BOB	Boliviano	591			es-BO,qu,ay
BQ	BES	535		Bonaire, Saint Eustatius and	Saba		18012	NA	North America	.bq	USD	Dollar	599			nl,pap,en
BR	BRA	76	BR	Brazil	Brasilia	8511965	201103330	SA	South America	.br	BRL	Real	55	#####-###	^(\d{8})$	pt-BR,es,en,fr
BS	BHS	44	BF	Bahamas	Nassau	13940	301790	NA	North America	.bs	BSD	Dollar	-241			en-BS
BT	BTN	64	BT	Bhutan	Thimphu	47000	699847	AS	Asia	.bt	BTN	Ngultrum	975			dz
BV	BVT	74	BV	Bouvet Island			0	AN	Antarctica	.bv	NOK	Krone				
BW	BWA	72	BC	Botswana	Gaborone	600370	2029307	AF	Africa	.bw	BWP	Pula	267			en-BW,tn-BW
BY	BLR	112	BO	Belarus	Minsk	207600	9685000	EU	Europe	.by	BYR	Ruble	375	######	^(\d{6})$	be,ru
BZ	BLZ	84	BH	Belize	Belmopan	22966	314522	NA	North America	.bz	BZD	Dollar	501			en-BZ,es
CA	CAN	124	CA	Canada	Ottawa	9984670	33679000	NA	North America	.ca	CAD	Dollar	1	@#@ #@#	^([ABCEGHJKLMNPRSTVXY]\d[ABCEGHJKLMNP	en-CA,fr-CA,iu
CC	CCK	166	CK	Cocos Islands	West Island	14	628	AS	Asia	.cc	AUD	Dollar	61			ms-CC,en
CD	COD	180	CG	Democratic Republic of the (	Kinshasa	2345410	70916439	AF	Africa	.cd	CDF	Franc	243			fr-CD,ln,kg
CF	CAF	140	CT	Central African Republic	Bangui	622984	4844927	AF	Africa	.cf	XAF	Franc	236			fr-CF,sg,ln,kg
CG	COG	178	CF	Republic of the Congo	Brazzaville	342000	3039126	AF	Africa	.cg	XAF	Franc	242			fr-CG,kg,ln-CG

Style Elements

If we filter on the `Continent` column we see there are 7 continents.



SQL SELECT Statement	Excel data file (.xlsx, .xlsm, .xlsb, or .xls only)	Status
`# Specify the workbook containing the data`		
`SET DATA FILE`	countries.xlsx	
`# Create a node for each country`		
`SELECT [ISO] as [Item], [ISO] as [Label], [Country] as [External Label], [Continent] as [Style Name] FROM [countries$] WHERE [ISO] IS NOT NULL ORDER BY [Country] ASC`		
`# Create a node for each continent`		
`SELECT DISTINCT [Continent] as [Item], [Continent] as [Label], "Continent" as [Style Name] FROM [countries$] ORDER BY [Continent] ASC`		
`# Create an edge between neighboring countries`		
`SELECT [ISO] as [Item], [Neighbors] as [Related Item] FROM [countries$] WHERE [Neighbors] IS NOT NULL`		
`# Create an edge between each country and its parent continent`		
`SELECT [Continent] as [Item], [ISO] as [Related Item], "ContinentToCountry" as [Style Name] FROM [countries$] WHERE [ISO] IS NOT NULL`		

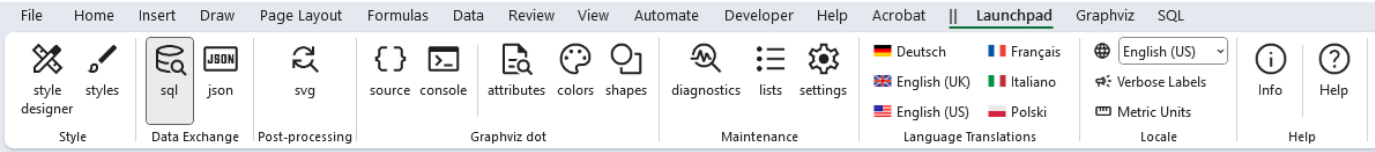
Using the Style Designer we create a unique style for each country by assigning different Fill Colors. The styles are stored on the styles worksheet using the Continent Name as the Style Name.

This snippet from the `styles` worksheet shows the 7 node style definitions which correspond to the continent names.

Africa	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="1" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	Africa
Antarctica	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="2" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	Antarctica
Asia	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="3" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	Asia
Europe	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="4" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	Europe
North America	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="5" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	North America
Oceana	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="6" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	Oceana
South America	shape="circle" height="0.375" width="0.375" fixedsize="shape" colorscheme="pastel27" fillcolor="7" fontname="Calibri" fontsize="10" style="filled"	node	yes	yes	yes	South America

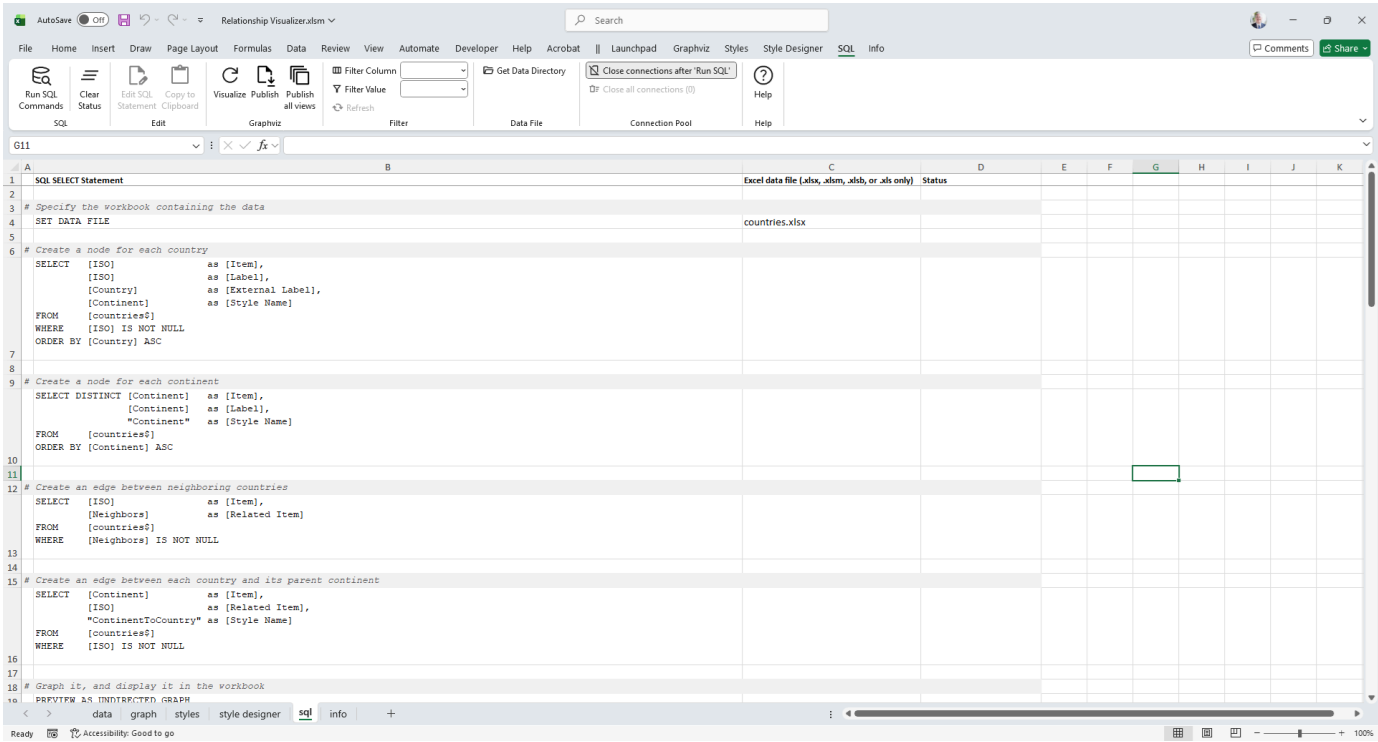
Building the SQL Queries and Directives

Next, open `Relationship Visualizer.xlsm` workbook. From the Launchpad ribbon tab select `SQL`.



The `sql` worksheet will appear. You'll rows containing SQL statements, and some additional directives which. These queries extract values from the `countries` workbook and are used to generate a graph showing continents, countries, and their shared borders.

The `sql` worksheet appears as follows:



Specify the Data Source

The first directive statement on row 4 is `SET DATA FILE`. It informs the Relationship Visualizer to use the file specified in column C (`countries.xlsx`) to be the default data source.

Create the Country Nodes

The first SQL statement on row 7 selects the column `ISO` to represent the `Item`, as well as the node `label`. The `Country` name is selected to be an `External Label`, and the `Continent` name is selected to be the `Style Name`.

The SQL is written as follows:

```sql
SELECT [ISO]       as [Item],
       [ISO]       as [Label],
       [Country]   as [External Label],
       [Continent] as [Style Name]
FROM  [countries$]
WHERE [ISO] IS NOT NULL
ORDER BY [Country] ASC
```

Create the Continent Nodes

The second SQL statement on row 10 will extract the 7 seven unique continent names from the list of countries by using the `DISTINCT` clause. The `Continent` name will be used as the `Item ID` as well as the node `label`. A `Style Name` of **Continent** will be used for all the rows.

The SQL is written as follows:

```sql
SELECT DISTINCT [Continent] as [Item],
                [Continent] as [Label],
                'Continent' as [Style Name]
FROM [countries$]
ORDER BY [Continent]
```

Create the Country-to-Country Edges

The third SQL statement on row 13 creates edge relationship rows by selecting the ISO value as the `Item ID`, and the `Neighbors` column as the `Related Item` value. The `Neighbors` column in the source worksheet contains comma-delimited `ISO` values which are neighboring countries to that row. The Relationship Visualizer has built-in logic to expand the comma-delimited list into multiple relationships. No style name is provided, so the default edge style will be used. Note that on the `WHERE` clause the directive `IS NOT NULL` has been added. This clause causes the query to skip rows with empty cells, as there are no values with which to express relationships.

The SQL is written as follows:

```sql
SELECT [ISO]       as [Item],
       [Neighbors] as [Related Item]
FROM   [countries$]
WHERE [Neighbors] IS NOT NULL
```

Create the Continent-to-Country Edges

The fourth SQL statement on row 16 is used to group countries by continent. It creates edge relationships by placing the Continent name as the `Item ID`, and the ISO country code in the `Related Item` column. The `Style Name` is specified as **ContinentToCountry** which has been defined as an invisible edge.

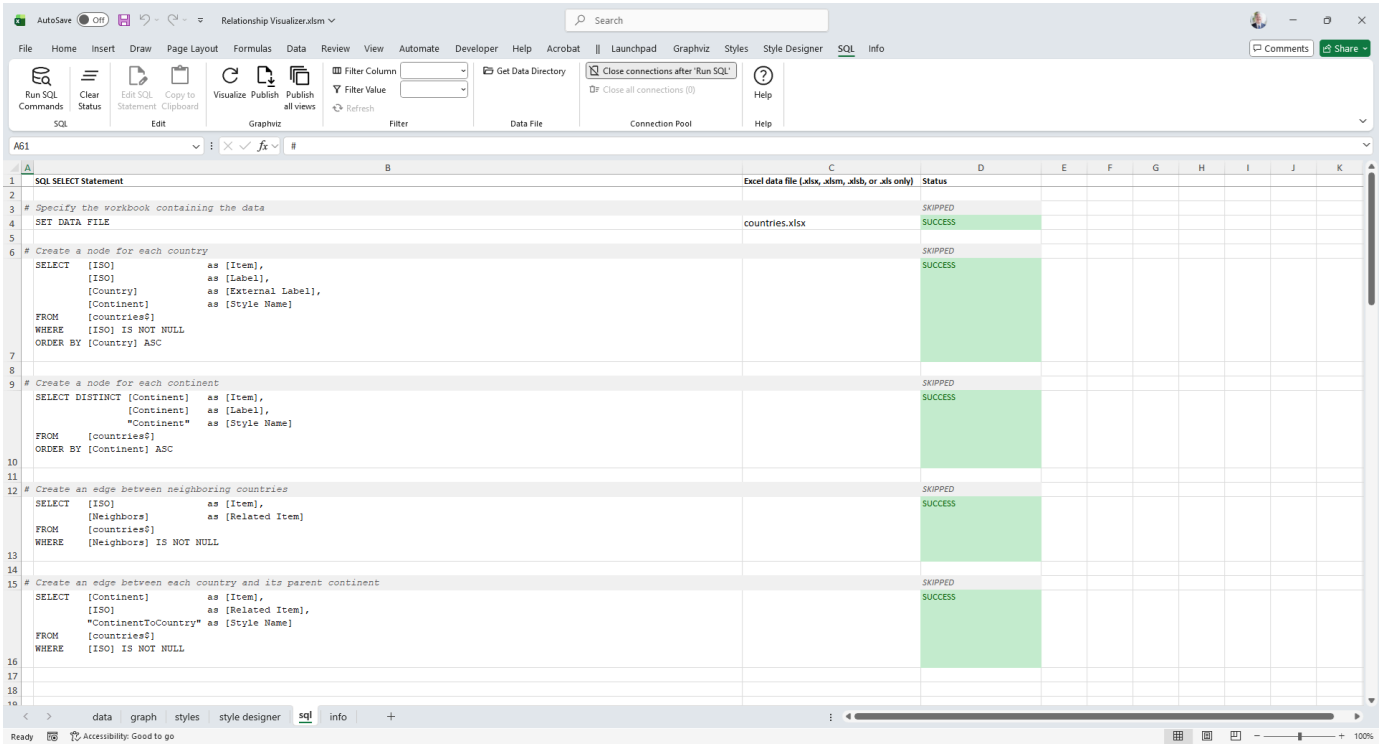
The SQL is written as follows:

```sql
SELECT [Continent]          as [Item],
       [ISO]                as [Related Item],
       'ContinentToCountry' as [Style Name]
FROM [countries$]
WHERE [ISO] IS NOT NULL
```

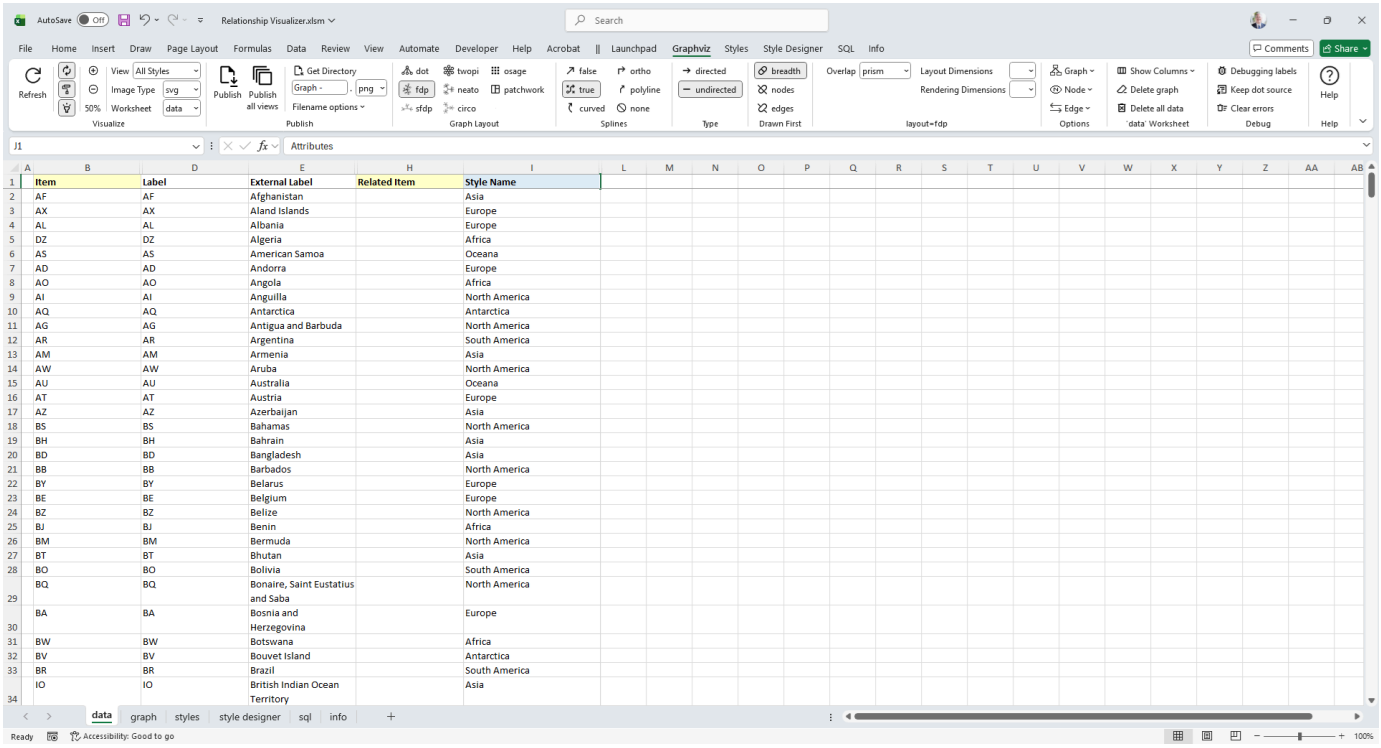
Run the SQL Queries

At this point our queries are complete. Press the `Run SQL Commands` button.

The SQL commands are run in sequence from top to bottom. Results are written to the `data` worksheet, and the query result status is displayed in column D such as:



If we switch worksheets to the `data` worksheet, it appears as follows:



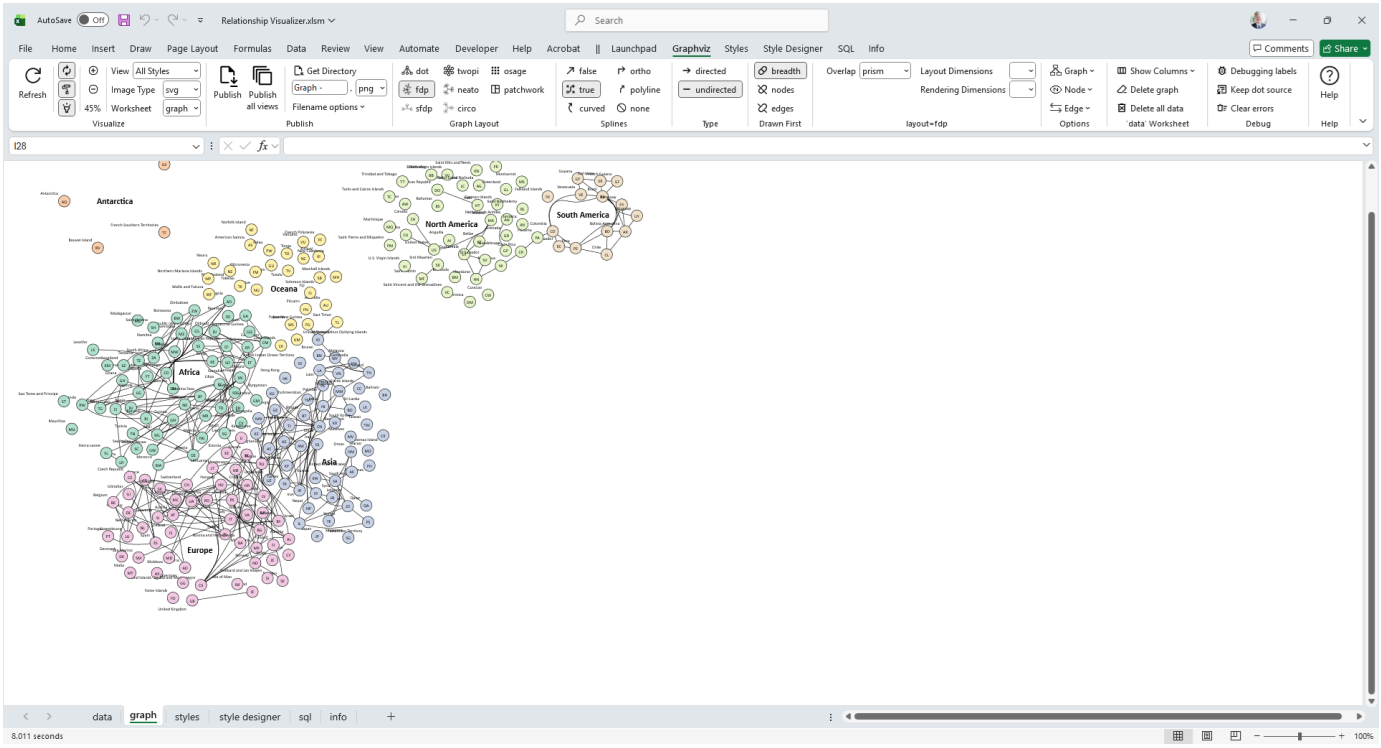
Item	Label	External Label	Related Item	Style Name
AF	AF	Afghanistan		Asia
AX	AX	Aland Islands		Europe
AL	AL	Albania		Europe
DZ	DZ	Algeria		Africa
AS	AS	American Samoa		Oceana
AD	AD	Andorra		Europe
AO	AO	Angola		Africa
AI	AI	Anguilla		North America
AQ	AQ	Antarctica		Antarctica
AG	AG	Antigua and Barbuda		North America
AR	AR	Argentina		South America
AM	AM	Armenia		Asia
AW	AW	Aruba		North America
AU	AU	Australia		Oceana
AT	AT	Austria		Europe
AZ	AZ	Azerbaijan		Asia
BS	BS	Bahamas		North America
BH	BH	Bahrain		Asia
BD	BD	Bangladesh		Asia
BB	BB	Barbados		North America
BY	BY	Belarus		Europe
BE	BE	Belgium		Europe
BZ	BZ	Belize		North America
BJ	BJ	Benin		Africa
BM	BM	Bermuda		North America
BT	BT	Bhutan		Asia
BO	BO	Bolivia		South America
BQ	BQ	Bonaire, Saint Eustatius and Saba		North America
BA	BA	Bosnia and Herzegovina		Europe
BW	BW	Botswana		Africa
BV	BV	Bouvet Island		Antarctica
BR	BR	Brazil		South America
IO	IO	British Indian Ocean Territory		Asia

The data is all present and in the appropriate columns for graphing. In this example **678 rows of data have been created using 4 SQL statements!**

Graph the Data

Press the `Refresh` button to graph the data. Since this is a large data set, be prepared to wait a little while for the results.

When Graphviz completes its work, you should see a logical world graph which appears as follows :



Options used: Worksheet = `graph` *, Zoom = 45%, layout=fdp, splines=true, graphtype=undirected.*

In a few short minutes we have gone from tabular Excel data to a graph visualization.

SQL Extensions

SQL extensions provide small, declarative utilities that simplify common graph-building tasks in the **Relationship Visualizer**.

Each extension is activated by passing specific values as SQL parameters, enabling expressive diagrams with minimal query logic.

Use the cards below to explore each extension.

Directives

Lightweight commands that enable optional behaviors within the SQL pipeline.

Group Nodes into Nested Clusters

Group related rows into clusters or subclusters to visually organize sections of your graph.

Split Long Label Text

Split long text labels into multiple lines for improved readability.

Align Nodes on the Same Level

Wrap selected nodes into ranked subgraphs to control layout, alignment, and visual grouping.

Generate Edges to Chain Nodes

Generate edges between sequential nodes to create simple chains—useful for timelines or ordered flows.

Traverse Trees Recursively

Use recursive SQL to walk hierarchical data and produce parent-child structures such as organization charts.

Iterate SQL Results

Iterate over SQL query results to

Enumerate Values

Assign incremental numbers to

execute a follow-up query using the initial results.

rows for ordering, labeling, or sequence-based logic.

Substitute Counts

Automatically substitute cluster, subcluster, and row counts into labels and sorting attributes.

Concatenate Values

Combine multiple fields or computed values into a single label, often used together with iteration.

Directives

The Relationship Visualizer has additional directives which look like SQL which can be used to enhance the batch processing capabilities. Think of them as extensions to SQL.

Clear the results from the `data` worksheet

- `RESET`

Specify the workbook containing the data

- `SET DATA FILE` where the workbook name is placed in Column C.

Placeholders

Set placeholder values for substitution into SQL, and wrap placeholder name in braces in the SQL.

- `SET PLACEHOLDER placeholder_name = value`

For example, to query for records matching the value "Metropolitan", assign Metropolitan to a placeholder, then use the placeholder name wrapped in braces in the SQL as static text in WHERE comparisons, label, or tooltip text.

```sql
SET PLACEHOLDER subway_line = Metropolitan

SELECT [station_from]       AS [Item],
       [station_to]         AS [Related Item],
       '{subway_line} Line' AS [Label],
```

```
FROM   [london_underground$]
WHERE  [tube_route] = '{subway_line}'
```

Show the graph immediately after all SQL statements processed

Pattern: `PREVIEW [ AS ( DIRECTED | UNDIRECTED ) GRAPH ]`

- `PREVIEW`
- `PREVIEW AS DIRECTED GRAPH`
- `PREVIEW AS UNDIRECTED GRAPH`

Publish the graphs as files

Pattern: `PUBLISH [ ALL VIEWS ] [ AS ( DIRECTED | UNDIRECTED ) GRAPH ] [ file prefix ]`

- `PUBLISH`
- `PUBLISH AS DIRECTED GRAPH`
- `PUBLISH AS UNDIRECTED GRAPH`
- `PUBLISH ALL VIEWS`
- `PUBLISH ALL VIEWS AS DIRECTED GRAPH`
- `PUBLISH ALL VIEWS AS UNDIRECTED GRAPH`

You can specify a value to use for the File Name Prefix as the last value of the directive. for example, if the desired prefix is `foobar`, the directives are:

- `PUBLISH foobar`
- `PUBLISH AS DIRECTED GRAPH foobar`
- `PUBLISH AS UNDIRECTED GRAPH foobar`
- `PUBLISH ALL VIEWS foobar`

- `PUBLISH ALL VIEWS AS DIRECTED GRAPH foobar`
- `PUBLISH ALL VIEWS AS UNDIRECTED GRAPH foobar`

Logging

Turn on/off logging of errors to file `Relationship Visualizer ADO Log.txt`

- `ENABLE LOGGING`
- `DISABLE LOGGING`

Environment Diagnostics

Write environment diagonostic information to file `Relationship Visualizer ADO Log.txt`

- `LOG ENVIRONMENT`

Count Substitution

Several substitution strings are available for inserting count values as clusters, subclusters, and records are iterated over. Before data is written to the `data` worksheet, a find-and-replace operation substitutes each placeholder with the current counter value. These counters are useful for preserving sort order or for appending values to style names so each cluster or subcluster can receive a distinct style.

Counter	Substitution token
Cluster	`{clc}`
Subcluster	`{scc}`
Result set record	`{rsc}`

These counts are especially helpful when you want to assign different border styles per cluster or subcluster, or when you need to emit a sort order using the `sortv` attribute. The `sortv` attribute is particularly valuable when using the [osage layout] to create heatmaps or domain models where controlling the order of columns—or the elements within each column—is important.

For example:

```sql
SELECT
    [Continent]       AS [CLUSTER],
    [Continent]       AS [CLUSTER LABEL],
    'Continent_{clc}' AS [CLUSTER STYLE NAME],

    [Region]          AS [SUBCLUSTER],
    [Region]          AS [SUBCLUSTER LABEL],
    'Region_{scc}'    AS [SUBCLUSTER STYLE NAME],

    [Country]          AS [ITEM],
    'sortv={rsc}'      AS [ATTRIBUTES]
```

```
FROM [Countries$]
ORDER BY [Continent], [Region], [Country]
```

This snippet illustrates:

- `{clc}` → increments once per cluster (continent)
- `{scc}` → increments once per subcluster (region)
- `{rsc}` → increments once per record in the result set

Example 1

The following examples show how the `{clc}`, `{scc}`, and `{rsc}` counters are substituted as the SQL iterates through clusters, subclusters, and individual records. These counters allow you to generate unique labels, styles, and sort orders directly from the query output.

Substitution tokens

There is nothing inherently special about the `{}` characters used in the substitution strings—you can change them to any token you prefer on the settings worksheet. The only requirement is that the token be something unlikely to appear in your data.

Example:
If you prefer more visually distinctive markers, you could change:

- `{clc}` → `<clc>`
- `{scc}` → `@scc@`
- `{rsc}` → `%rsc%`

Your SQL would then use these new tokens, and the substitution engine will replace them exactly the same way as the defaults.

Record, Cluster, and Subcluster Counters

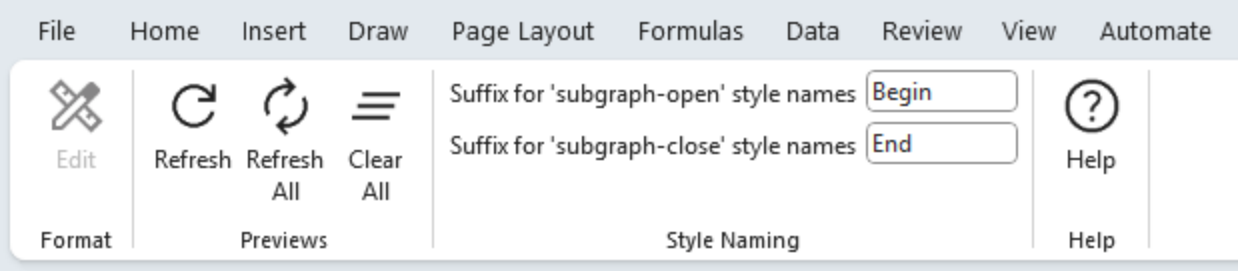
Row	Continent	Region	Country	{clc}	{scc}	{rsc}	Cluster Label	Su La
1	Africa	East Africa	Kenya	1	1	1	Africa	Ea
2	Africa	East Africa	Uganda	1	1	2	Africa	Ea
3	Africa	West Africa	Ghana	1	2	3	Africa	W Af
4	Europe	Northern Europe	Sweden	2	1	4	Europe	N Eu
5	Europe	Northern Europe	Norway	2	1	5	Europe	N Eu

Style Name Substitution

Row	Continent	Region	{clc}	{scc}	Cluster Style Name	Subcluster Style Name
1	Africa	East Africa	1	1	Continent_1	Region_1
2	Africa	East Africa	1	1	Continent_1	Region_1
3	Africa	West Africa	1	2	Continent_1	Region_2
4	Europe	Northern Europe	2	1	Continent_2	Region_1
5	Europe	Northern Europe	2	1	Continent_2	Region_1

We would need to create 4 cluster style definitions based on these results.

The cluster and subcluster style names will be modified at run-time to end with a begin/end suffix. The suffix values are configurable via the `Styles` ribbon tab.



The `Style Name` names needed are `Continent_1 Begin`, `Continent_2 Begin`, `Region_1 Begin`, `Region_2 Begin`, with matching paired style names to close the cluster, i.e. `Continent_1 End`, `Continent_2 End`, `Region_1 End`, `Region_2 End`, with matching

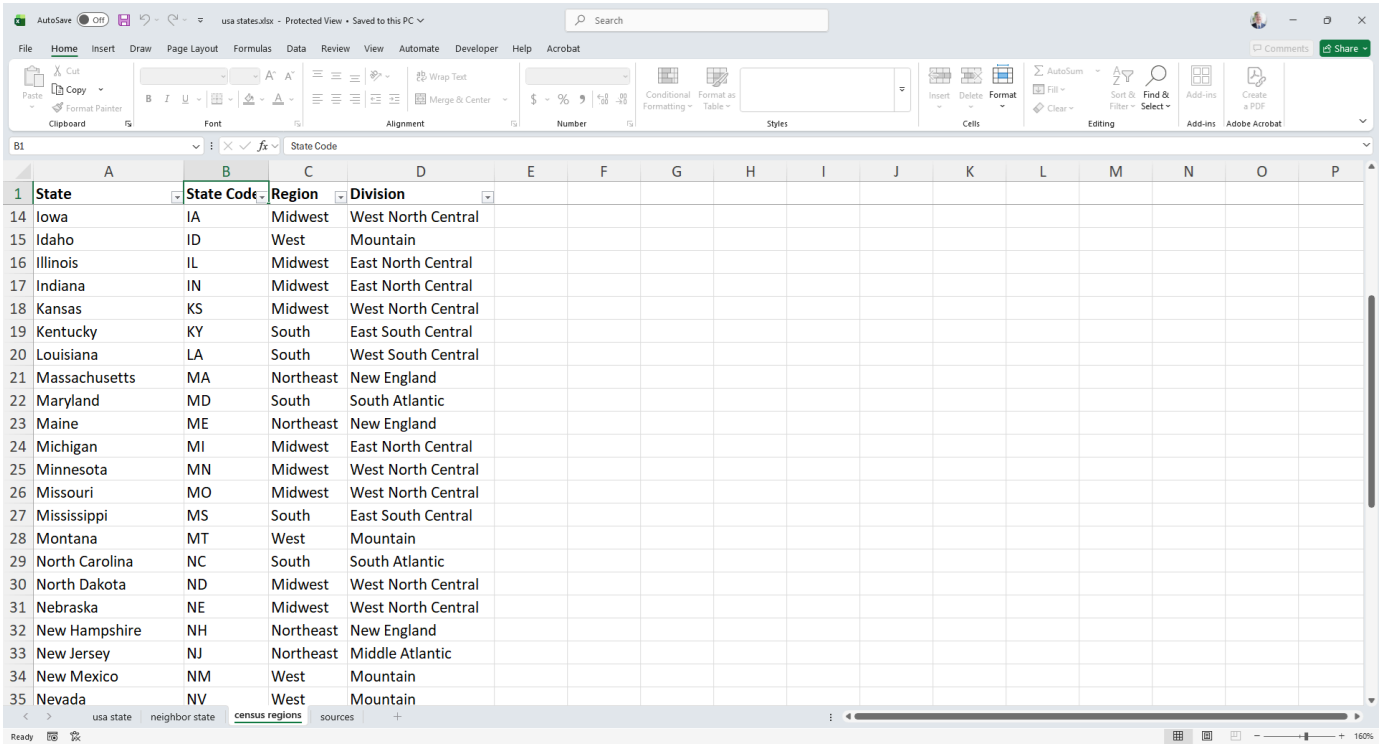
Remember to include that trailing space if you are using the five pre-defined borders on the styles worksheet.

These counters are especially powerful when generating heatmaps, osage layouts, or any diagram that relies on controlled ordering or dynamic styling. By embedding `{clc}`, `{scc}`, and `{rsc}` directly into labels, style names, or attributes, you can assign unique visual treatments to each cluster or subcluster and enforce a predictable sort order. This makes it easy to highlight categories, create graded color schemes, or arrange elements consistently across multiple graph views.

Example 2

Use US census information to depict the 50 US states, grouping the states by by region of the country, and census division. State names should be in alphabetical order.

This is the Excel worksheet to graph:



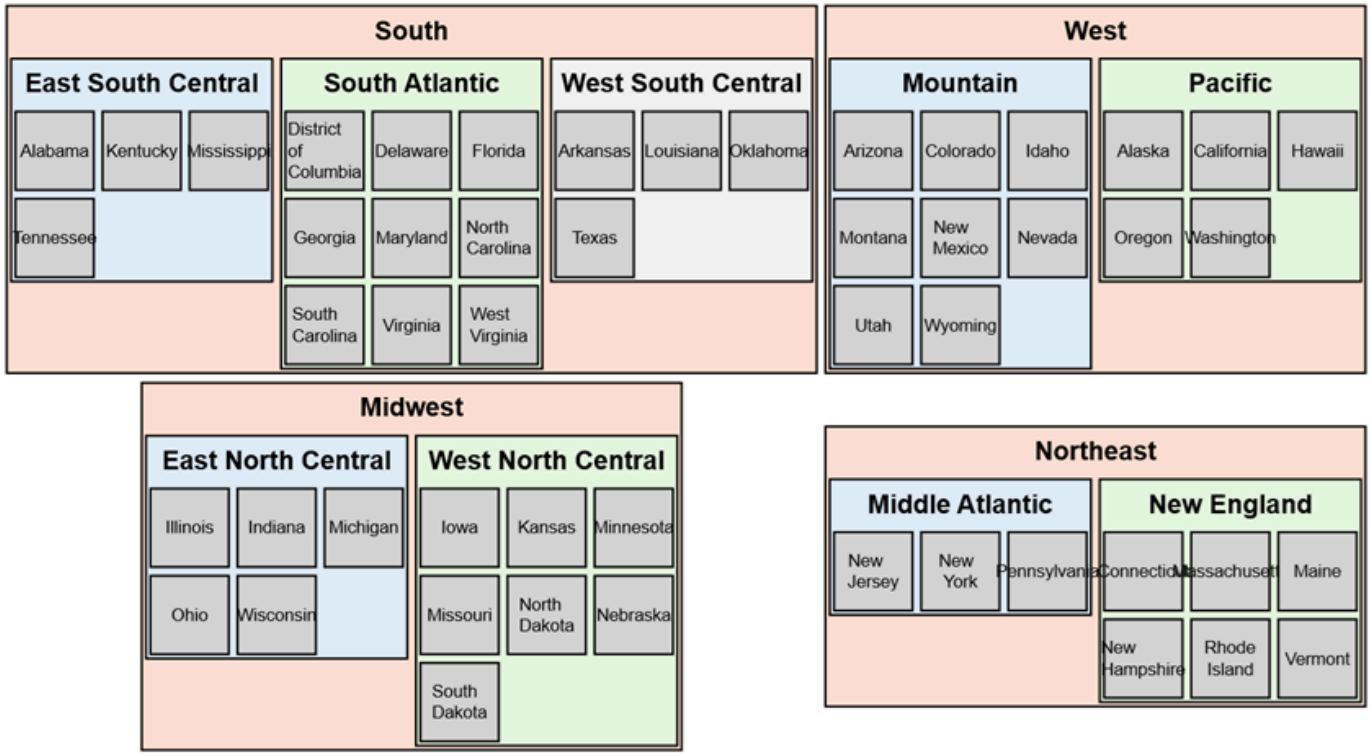
	A	B	C	D
1	State	State Code	Region	Division
14	Iowa	IA	Midwest	West North Central
15	Idaho	ID	West	Mountain
16	Illinois	IL	Midwest	East North Central
17	Indiana	IN	Midwest	East North Central
18	Kansas	KS	Midwest	West North Central
19	Kentucky	KY	South	East South Central
20	Louisiana	LA	South	West South Central
21	Massachusetts	MA	Northeast	New England
22	Maryland	MD	South	South Atlantic
23	Maine	ME	Northeast	New England
24	Michigan	MI	Midwest	East North Central
25	Minnesota	MN	Midwest	West North Central
26	Missouri	MO	Midwest	West North Central
27	Mississippi	MS	South	East South Central
28	Montana	MT	West	Mountain
29	North Carolina	NC	South	South Atlantic
30	North Dakota	ND	Midwest	West North Central
31	Nebraska	NE	Midwest	West North Central
32	New Hampshire	NH	Northeast	New England
33	New Jersey	NJ	Northeast	Middle Atlantic
34	New Mexico	NM	West	Mountain
35	Nevada	NV	West	Mountain

This SQL statement will process the requirements. It combines clusters, subclusters, split text and count substitution.

```sql
SELECT
  [State Code]                        as [item],
  'Medium Square'                     as [style name],
  'sortv={rsc}'                       as [attributes],
  [State]                             as [label],
  5                                   as [split length],
  '\l'                                as [line ending],
  [State Code]                        as [external label],
  [State]                             as [tooltip],
  [Region]                            as [cluster],
  [Region]                            as [cluster label],
  'Border 6 '                         as [cluster style name],
  [Region]                            as [cluster tooltip],
  'sortv={clc} packmode=array_utr3'   as [cluster attributes],
  [Division]                          as [subcluster],
  [Division]                          as [subcluster label],
  'Border {scc} '                     as [subcluster style name],
  [Division]                          as [subcluster tooltip],
  'sortv={scc} packmode=array_utr3'   as [subcluster attributes]
```

```
FROM
  [census regions$]
ORDER BY
  [Region]     ASC,
  [Division]   ASC,
  [State Code] ASC
```

This is the graph produced by the data plus the SQL query.



Try it Yourself

This example is included in the samples in the Relationship Visualizer zip file in the directory `06 - Using SQL - Clusters and Subclusters`.

Split Long Labels

Label splitting is a powerful SQL extension that allows you to break long label text into multiple readable lines within a node. It gives you control over where lines break and how each line is aligned (left, center, or right), resulting in cleaner, more balanced, and professional-looking diagrams.

When to Use Label Splitting

- Long titles or descriptions that make nodes too wide
- Improving readability in dense diagrams
- Creating consistent node widths across different amounts of text

How It Works

Add two columns to your SQL query

Column Name	Description
`SPLIT LENGTH`	Desired maximum characters per line
`LINE ENDING`	Line break character and alignment

as follows:

```sql
        '5'  as [SPLIT LENGTH],
        '\n' as [LINE ENDING],
```

In this example, the label will be split into multiple lines at boundaries as close as possible to 12 characters. Splits occur only at spaces, so any word longer than 12 characters will remain unbroken.

Line endings can be any string[1]. The most commonly used line endings are:

Line Ending	Meaning / Usage
`\n`	New line with center alignment
`\r`	New line with right alignment
`\l`	New line with left alignment
`|`	Pipe delimiter (useful for Record shapes)
` `	HTML line break (for HTML-like labels)

[1] New line `\n` is the default if `SPLIT LENGTH` is specified, but `LINE ENDING` is omitted.

Here is an example:

Before:

This is a very long label that stretches the node

After:

This is a very long label that stretches the node

Enumerate Values

Enumerated SQL queries enable a query to loop through a numeric range and generate rows for each value in that range. Enumeration is especially useful when you need to fill in missing values, create evenly spaced steps, or build a complete sequence that does not exist in the underlying data.

Iterate vs. Enumerate?

Within Relationship Visualizer, a distinction is made between **iteration** and **enumeration**:

- **Iterate** refers to stepping through each item in a result set one by one — without tracking position or count. Each item becomes input for a follow-up SQL query.
- **Enumerate** refers to iteration **with explicit tracking of position, index, or count**, using a counter that increments from a starting value to a stopping value. Enumeration is used when the logic depends on the numeric sequence itself. For example, "generate rows from 1 to 12," "step through values by 5," or "fill in missing years."

Enumeration is not about iterating over data — it's about generating **new** data based on a numeric range.

Problem Scenario

In the Iteration Queries example we built a timeline of Unix shell introductions. The dataset contains a `Year` column, but the years are sparse:

- Some years have shells.
- Many years do not.

The following query creates edges from one year to the next **only for years that appear in the data**:

```sql
SELECT DISTINCT
       ([Year] & '_id') AS [ITEM],
       TRUE AS [CREATE EDGES],
       [Year]
FROM [Shells$]
WHERE [Year] IS NOT NULL
ORDER BY [Year] ASC
```

If we create edges directly from the data, the timeline will “jump” over missing years, compressing the visualization and making the evolution appear faster than it actually was.

Fixing the Gaps with Enumeration

Relationship Visualizer supports **enumerated SQL queries** through four special fields: `ENUMERATE`, `START AT`, `STOP AT`, and `STEP BY`. When these appear in the `SELECT` list, the engine switches into enumeration mode and generates a numeric sequence independent of any table data.

Enumeration is the correct tool when you need a **complete, gap-free range** such as years, months, sequence numbers, or evenly spaced steps. Instead of relying on whatever values happen to exist in the dataset, enumeration produces a deterministic backbone that fills in all missing points.

Basic Enumeration Syntax

The simplest enumerated query defines:

- `TRUE AS [ENUMERATE]` - Indicates that enumeration is desires/active
- `... AS [START AT]` - The starting value
- `... AS [STOP AT]` - The **inclusive** stopping value

- `... AS [STEP BY]` - The increment
- How each generated step should appear in the output.

  Within any text field, the placeholder `{step}` is replaced with the current numeric value. This allows you to generate IDs, labels, keys, or synthetic attributes without referencing any underlying table.

The result is one row per step, covering the entire numeric range you specify.

Examples

Example 1: Create the nodes from year to year using a hard-coded range

Enumeration does not depend on any table — it **creates** data. You can then join, merge, or visualize this generated sequence alongside your real dataset.

```sql
SELECT TRUE AS [ENUMERATE],
       1965 AS [START AT],
       1995 AS [STOP AT],
       1 AS [STEP BY],
       '{step}' AS [Item],
       '{step}' AS [Label],
       'Year'   AS [Style Name]
```

This SQL creates the nodes, and applies a style named **Year** from the `styles` worksheet to the node which is created.

Example 2: Create the edges from year to year using a hard-coded range

This example shows how to generate a complete sequence of years using a fixed numeric range. By enabling both `CREATE EDGES` and `ENUMERATE`, the engine produces one row per year and automatically creates edges between consecutive values.

Because the range is hard-coded, the query does not depend on any underlying table. It simply emits a synthetic timeline from the starting year up to (but not including) the stopping year. Each iteration replaces `{step}` with the current year, allowing you to generate IDs, labels, or edge endpoints directly from the loop counter.

This pattern is useful when you want a predictable, fully populated timeline regardless of what data exists in the workbook.

```sql
SELECT TRUE AS [CREATE EDGES],
       TRUE AS [ENUMERATE],
       1965 AS [START AT],
       1995 AS [STOP AT],
       1 AS [STEP BY],
       '{step}' AS [Item]
```

Example 3: Create nodes for the minimum year through the maximum year

This example shows how to generate a complete sequence of years based on the actual data in the workbook. Instead of hard-coding the range, the query calculates the minimum and maximum year values directly from the table and uses them as the bounds for enumeration.

By doing this, the generated sequence automatically adapts to whatever data is present. If new shells are added with earlier or later years, the enumerated range expands accordingly. The result is a flexible, data-driven timeline that always covers the full span of years represented in the dataset.

Each iteration replaces `{step}` with the current year, producing one node per year. These nodes can be used as standalone timeline markers or combined with edges to create a continuous, gap-free visualization of the entire period.

```sql
SELECT TRUE AS [ENUMERATE],
       MIN(CLng([Year])) AS [START AT],
       MAX(CLng([Year])) AS [STOP AT],
       1 AS [STEP BY],
```

```
        '{step}' AS [Item],
        '{step}' AS [Label]FROM [shells$]
WHERE IsNumeric([Year])
```

Example 4: Create edges for the minimum year through the maximum year

This example combines data-driven bounds with automatic edge creation. By enabling both `ENUMERATE` and `CREATE EDGES`, the engine generates a continuous sequence of years based on the minimum and maximum values found in the dataset, and then creates edges between each consecutive pair.

Because the range is derived from the actual data, the resulting timeline always expands or contracts to match the earliest and latest years present in the table. This makes the query fully adaptive: adding new rows with earlier or later years automatically updates the generated edges.

Each iteration substitutes `{step}` with the current year, producing a clean, gap-free chain of edges that spans the entire period represented in the dataset. This is the most flexible way to build a complete chronological backbone for visualization.

```sql
SELECT TRUE AS [CREATE EDGES],
       TRUE AS [ENUMERATE],
       MIN(CLng([Year])) AS [START AT],
       MAX(CLng([Year])) AS [STOP AT],
       1 AS [STEP BY],
       '{step}' AS [Item]
FROM [shells$]
WHERE IsNumeric([Year])
```

Try it Yourself

This example is included in the samples in the Relationship Visualizer zip file in the directory `17 - Using SQL - Enumeration`.

Summary

Enumeration gives Relationship Visualizer a way to generate data that does not exist in any table, making it possible to build complete sequences, fill gaps, and create smooth visual structures such as timelines. By defining a numeric range with `START AT`, `STOP AT`, and `STEP BY`, you gain full control over how many rows are produced and how they are labeled. This makes enumeration ideal for scaffolding, synthetic nodes, and any situation where the visualization depends on a continuous progression of values.

Iteration and enumeration complement each other: iteration processes what the data already contains, while enumeration creates what the data is missing. Used together, they allow you to combine real and synthetic information into a single, coherent model that supports clearer, more expressive visualizations.

Concatenating Records into a Single Entry

We've shown how iteration identifies distinct values in a dataset and then loops through those values to run a second SQL statement that generates a subgroup for each one. **Concatenation** extends this pattern by combining the iterated values into a single string that can be used directly as a node label. Instead of emitting one row per item, you can aggregate related values—such as countries in a continent, members of a team, or songs on a playlist—into a single, well-structured label.

This example demonstrates how to add concatenation to an iteration query so that grouped data can be presented as a unified entry in your diagram.

Country Data

There is a workbook named **countries** in the sample directory. It contains a list of all countries around the world along with their parent continent.

The data is as follows:

"countries" Worksheet

Country	ContinentCode	Continent
Andorra	EU	Europe
United Arab Emirates	AS	Asia
Afghanistan	AS	Asia
Antigua and Barbuda	NA	North America
Anguilla	NA	North America
Albania	EU	Europe

Country	ContinentCode	Continent
Armenia	AS	Asia
Angola	AF	Africa
Antarctica	AN	Antarctica
Argentina	SA	South America
American Samoa	OC	Oceana
Austria	EU	Europe
Australia	OC	Oceana
etc. ...	...	...

Examples

The following examples illustrate two different ways to apply concatenation within an iteration query, each highlighting a distinct labeling style.

Example 1 - `Rectangle` Shape Nodes

Iteration queries are passed the 2 SQL statements using the `SQL FOR ID` and `SQL FOR DATA` mappings.

The `SQL FOR ID` query first retrieves the distinct continent codes. It then iterates through those codes, running a secondary `SQL FOR DATA` query for each one to fetch all country names associated with that continent.

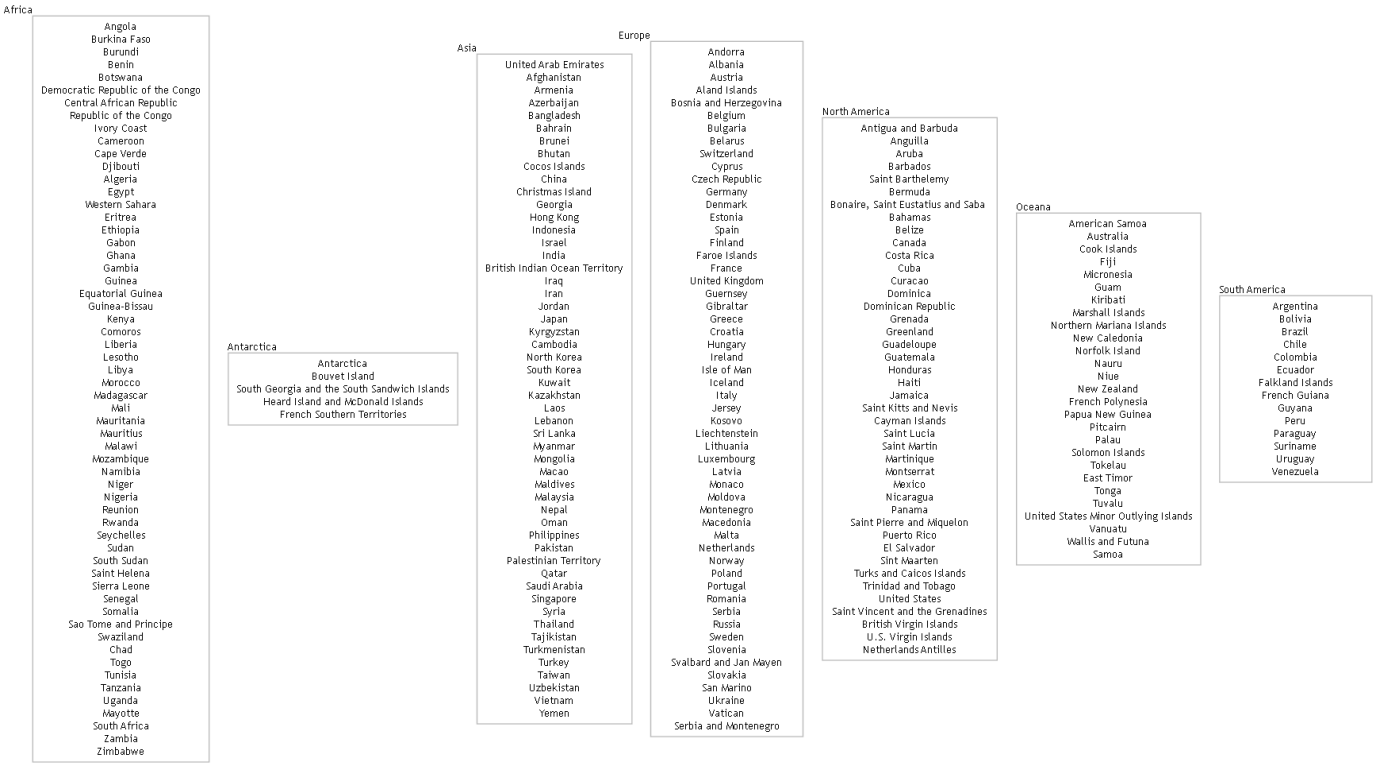
Concatenation is activated by the `TRUE AS [CONCATENATE]` value in the query. The country names get concatenated into a single node label, with each name separated by a newline (`\n`) so that every country appears on its own line within the rectangle. The concatenated

value is then assigned to the `[Label]` field using the `ASSIGN TO` setting, overriding the label value that would otherwise have been returned by the `SQL FOR ID` query.

Each emitted row in the `data` worksheet represents a single continent and is styled with the `Rectangle` style, which applies `shape="rectangle"`.

```sql
SELECT
  TRUE                                                              AS [ITERATE],
  'SELECT DISTINCT [ContinentCode] AS [ID], [continent] AS [Item], [Continent] AS [Exte
                                                                    AS [SQL FOR ID],
  'SELECT [Country] AS [Label] FROM [countries$] WHERE [ContinentCode] = ''{ID}'' AND
                                                                    AS [SQL FOR DATA],
  TRUE                                                              AS [CONCATENATE],
  'Label'                                                           AS [FIELD],
  'Label'                                                           AS [ASSIGN TO],
  ''                                                                AS [PREFIX],
  ''                                                                AS [SUFFIX],
  '\n'                                                              AS [SEPARATOR]
```

Since there are seven global continents, the resulting diagram contains seven rectangle-shaped nodes—one for each continent—each displaying its list of countries on separate lines. This produces a clear, vertically structured layout that makes it easy to compare continents at a glance:



Example 2 - `Record` Shape Nodes

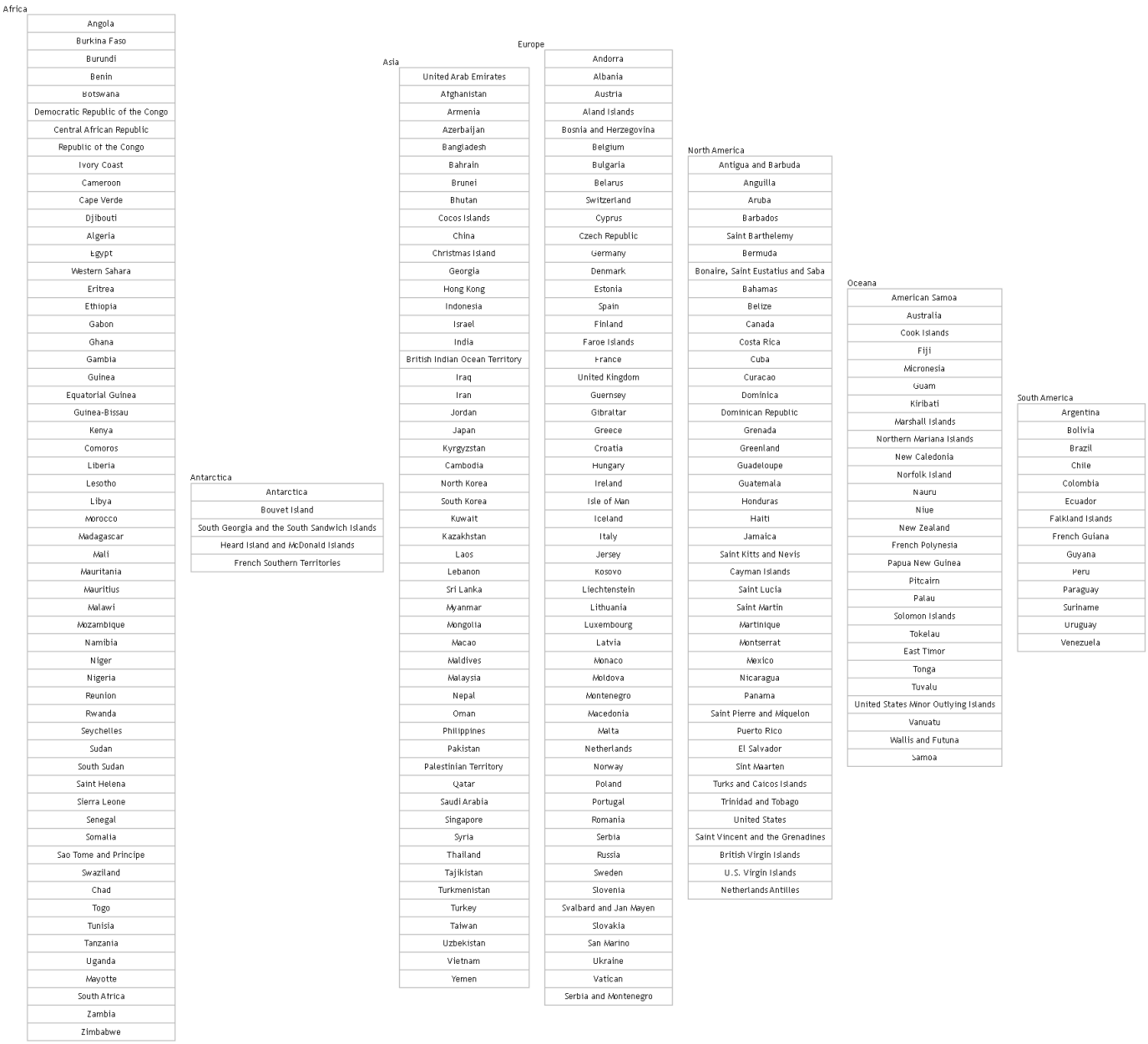
The `SQL FOR ID` query first retrieves the distinct continent codes. It then iterates through those codes, running a secondary `SQL FOR DATA` query for each one to fetch all country names associated with that continent. Each emitted row in the `data` worksheet represents a single continent and is styled with the `Record` style, which applies `shape="record"`.

Record shapes use a simple brace-and-pipe syntax (`{ }` and `|`) to define multi-row labels: the outer braces enclose the entire record, and each pipe character (`|`) separates one row (or field) from the next.

The country names are concatenated into a single record label row, separated by the pipe (`|`) character so that each name appears on its own line within the record. For this query, we pass `{` as the concatenation `PREFIX`, `|` as the `SEPARATOR`, and `}` as the `SUFFIX`, producing a properly formed record block for the label. The concatenated value is then assigned to the `[Label]` field using the `ASSIGN TO` setting, overriding the label value that would otherwise have been returned by the `SQL FOR ID` query.

```sql
SELECT
  TRUE                                                    AS [ITERATE],
  'SELECT DISTINCT [ContinentCode] AS [ID], [continent] AS [Item], [Continent] AS [Exte
                                                          AS [SQL FOR ID],
  'SELECT [Country] AS [Label] FROM [countries$] WHERE [ContinentCode] = ''{ID}'' AND
                                                          AS [SQL FOR DATA],
  TRUE                                                    AS [CONCATENATE],
  'Label'                                                 AS [FIELD],
  'Label'                                                 AS [ASSIGN TO],
  '{'                                                     AS [PREFIX],
  '}'                                                     AS [SUFFIX],
  '|'                                                     AS [SEPARATOR]
```

Since there are seven global continents, the resulting diagram contains seven record-shaped nodes—one for each continent—each displaying its list of countries as separate fields within the record. Because record shapes use the brace-and-pipe syntax to define rows, the concatenated country names appear as distinct segments inside the `{ … }` block, producing a clean, vertically structured layout that makes it easy to compare continents at a glance:



Summary

This sample demonstrates how concatenation can be combined with SQL iteration to produce clear, information-rich node labels from grouped data. By aggregating multiple rows into a single formatted string, you can present complex relationships—such as countries within a continent—in a compact, visually meaningful way. Whether rendered as record shapes or simple rectangles, concatenated labels allow your diagrams to

communicate structure at a glance while keeping the underlying SQL logic straightforward and reusable.

Sample Content

The files used in these examples are contained in the `\Relationship Visualizer\samples\19 - Using SQL - Concatenation` directory in the zip file download.

Grouping Data into Clusters and Subclusters

One extension adds the ability to scan SQL statements for specific field names that signal grouping values to be wrapped into a **cluster** or **subcluster** (a cluster within a cluster). The syntax remains pure SQL, but certain field names are reserved and given special meaning.

Clusters and subclusters can specify:

- A mandatory *cluster column*. The presence of this column triggers the additional processing needed to insert cluster braces into the result set.
- A **Label**, which allows you to override the label derived from the cluster column. If no label is provided, the value of the cluster column is used.
- A **Style Name**, which must correspond to a style defined on the `styles` worksheet. This can be:
  - a static string
  - a composite string using substitution or concatenation
  - or a column value that matches a style name
    A common example appears in heat-map scenarios, where values such as `Critical` and `Standard` map to styles of the same name, each with its own fill color.
- **Attributes**, which allow you to include line-specific attribute values.
- **Tooltip**, which provides hover text in SVG output.

Although the examples below use uppercase for clarity, the implementation is case-insensitive. Variants such as `CLUSTER`, `Cluster`, `ClUsTeR`, and `cluster` are all treated identically.

Column	Cluster Field Names	Subcluster Field Names
cluster column (i.e., group by)	`CLUSTER`	`SUBCLUSTER`
Label	`CLUSTER LABEL`	`SUBCLUSTER LABEL`
Style Name	`CLUSTER STYLE NAME`	`SUBCLUSTER STYLE NAME`

Column	Cluster Field Names	Subcluster Field Names
Attributes	`CLUSTER ATTRIBUTES`	`SUBCLUSTER ATTRIBUTES`
Tooltip	`CLUSTER TOOLTIP`	`SUBCLUSTER TOOLTIP`

Align Nodes on the Same Level

Ranking is useful whenever you want certain nodes to align visually in a consistent row or column. By placing nodes into a shared subgraph with a defined rank, you can control the layout and make structural relationships easier to understand. Common scenarios include:

- **Grouping Peers:** Aligning team members, departments, or sibling categories on the same horizontal level to emphasize equal standing.
- **Highlighting Stages:** Showing phases of a project or lifecycle (e.g., Planning, Execution, Review) on a single rank for clarity.
- **Organizing Layers:** Keeping all “input” nodes at the top, “processing” nodes in the middle, and “output” nodes at the bottom.
- **Comparing Alternatives:** Placing multiple options or branches side-by-side so users can visually compare them.
- **Clarifying Hierarchies:** Ensuring that children of the same parent appear on the same rank, preventing uneven or zig-zag layouts.
- **Creating Swimlanes:** Using ranks to form horizontal lanes that separate categories, roles, or functional areas.
- **Stabilizing Layouts:** When Graphviz’s automatic layout shifts nodes unpredictably, explicit ranks help lock the structure into a predictable shape.

These scenarios benefit from the `CREATE RANK` extension because it gives you precise control over how nodes align, making your graphs cleaner, more readable, and more intentional.

Assume you have an Excel workbook containing a worksheet named `Alphabet` with a column heading `letter` and four rows of data: A, B, C, and D. You want all of these nodes to appear on the same rank.

The **CREATE RANK** SQL extension produces subgraphs whose nodes share the same rank.

The SQL is specifed as follows:

```sql
SELECT [letter] AS [Item],
       TRUE     AS [CREATE RANK],
       'same'   AS [RANK]
FROM [Alphabet$]
```

The SQL above results in one row added to the `data` worksheet with:

- Item = `>`
- Label = `{rank="same"; "A"; "B"; "C"; "D";}`

The values you can pair with `RANK` are `same`, `min`, `source`, `max`, and `sink`. The Graphviz `rankdir` attribute appears to treat these values as case-sensitive, so be sure to specify them in **lowercase**.

Chain Nodes Using Edges

Chaining nodes is especially useful whenever your data represents an ordered sequence. Even a simple column of values can become a meaningful visual path once edges are created automatically. Common scenarios include:

- **Timelines:** Turning a list of dates or milestones into a left-to-right sequence that shows how events unfold.
- **Workflows and Processes:** Visualizing approval steps, onboarding stages, or any linear process where one step leads to the next.
- **Pipelines:** Representing ETL stages, data transformations, or processing phases in the order they occur.
- **Queues or Ordered Lists:** Showing the exact order in which tasks, tickets, or operations are handled.
- **Version Progression:** Displaying how versions evolve over time, such as `v1.0 → v1.1 → v1.2 → v2.0`.
- **Routes or Paths:** Mapping a sequence of locations, checkpoints, or waypoints into a clear path diagram.
- **Dependency Chains:** Illustrating simple “A must happen before B” relationships without needing a full dependency tree.
- **Learning or Reading Paths:** Presenting a recommended sequence of topics, lessons, or readings.

These scenarios all benefit from the automatic edge generation provided by `CREATE EDGES`, allowing you to transform a basic list into a structured, easy-to-read graph with minimal SQL.

Assume you have an Excel workbook with a worksheet named `Alphabet` that contains a column called `letter` with four rows of data: A, B, C, and D. Your goal is to chain these values together in sequence.

The following SQL creates nodes `A`, `B`, `C`, `D`:

```sql
SELECT [letter] AS [Item] from [Alphabet$]
```

The **CREATE EDGES** SQL extension automatically generates edges like `A -> B`, `B -> C`, and `C -> D`.

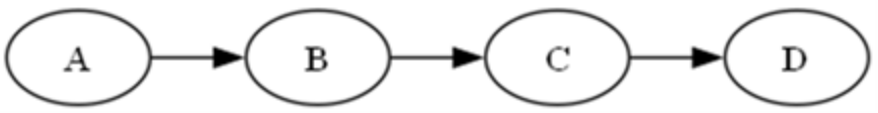
The SQL is specifed as follows:

```sql
SELECT DISTINCT [letter] AS [Item],
       TRUE              AS [CREATE EDGES]
FROM [Alphabet$]
ORDER BY [letter] ASC
```

The `DOT` source code appears as:

```
strict digraph "Relationship Visualizer"
{
    rankdir=LR;
    A -> B;
    B -> C;
    C -> D;
}
```

Producing this graph:



Traverse Trees Recursively

A frequent question is: *"Can you create tree-traversal graphs using the Relationship Visualizer?"*
The answer is **YES!**

Relationship Visualizer includes full support for recursive SQL queries, allowing it to automatically assemble hierarchical structures such as organization charts, dependency trees, and connected data paths. By repeatedly following parent–child relationships, the tool can expand even the simplest row-based dataset into a complete, multi-level hierarchy.

What is Recursion?

Recursion works by having a function repeatedly call itself, each time handling a smaller piece of the problem. The process continues until it reaches a base case that tells it when to stop.

Recursive SQL Queries

To use this feature effectively, the candidate dataset must include the key columns that define each relationship:

- A **parent** node
- A **child** node

These columns form the backbone of the hierarchy, allowing the recursive query engine to traverse upward or downward through the structure and generate a complete, layered graph.

A recursive query consists of two essential parts:

1. **Anchor the Base Case**
   This defines the starting point for recursion—typically the top-level parent in the hierarchy (for example, the CEO in an organization chart or the root node in a

dependency tree). The anchor member returns the initial set of rows from which the hierarchy will begin.

2. **Define the Recursive Member**
   This portion of the query describes how to repeatedly join the growing result set back to the source table. Each iteration locates the next level of child nodes, allowing the hierarchy to expand downward (or upward) until no additional relationships remain.

SQL Extensions

Four new keywords unique to Relationship Visualizer have been defined to enable recursive queries:

- `TREE QUERY` - Passed as a literal string, the presence of `TREE QUERY` in the SQL result set tells the SQL engine to execute the associated SQL recursively.

  The SQL **must** contain the symbolic value `''{WHERE VALUE}''` which will be replaced as each node is traversed with the value of the current node.

  Note that since `TREE QUERY` is passed as a literal string, any strings in the SQL which would normally be specified with `'` delimiters must be escaped as `''`.

- `WHERE VALUE` provides the value which anchors the base case. For example, in an organization chart it could be the ID of the CEO. For a subway line, it could be the starting subway station.

- `WHERE COLUMN` - Specifies the name of the column which defines the recursive member. It is the name of the column containing the value which should be searched next. For example, in an organization chart it could be column name of the column containing the ID of the employee's manager.

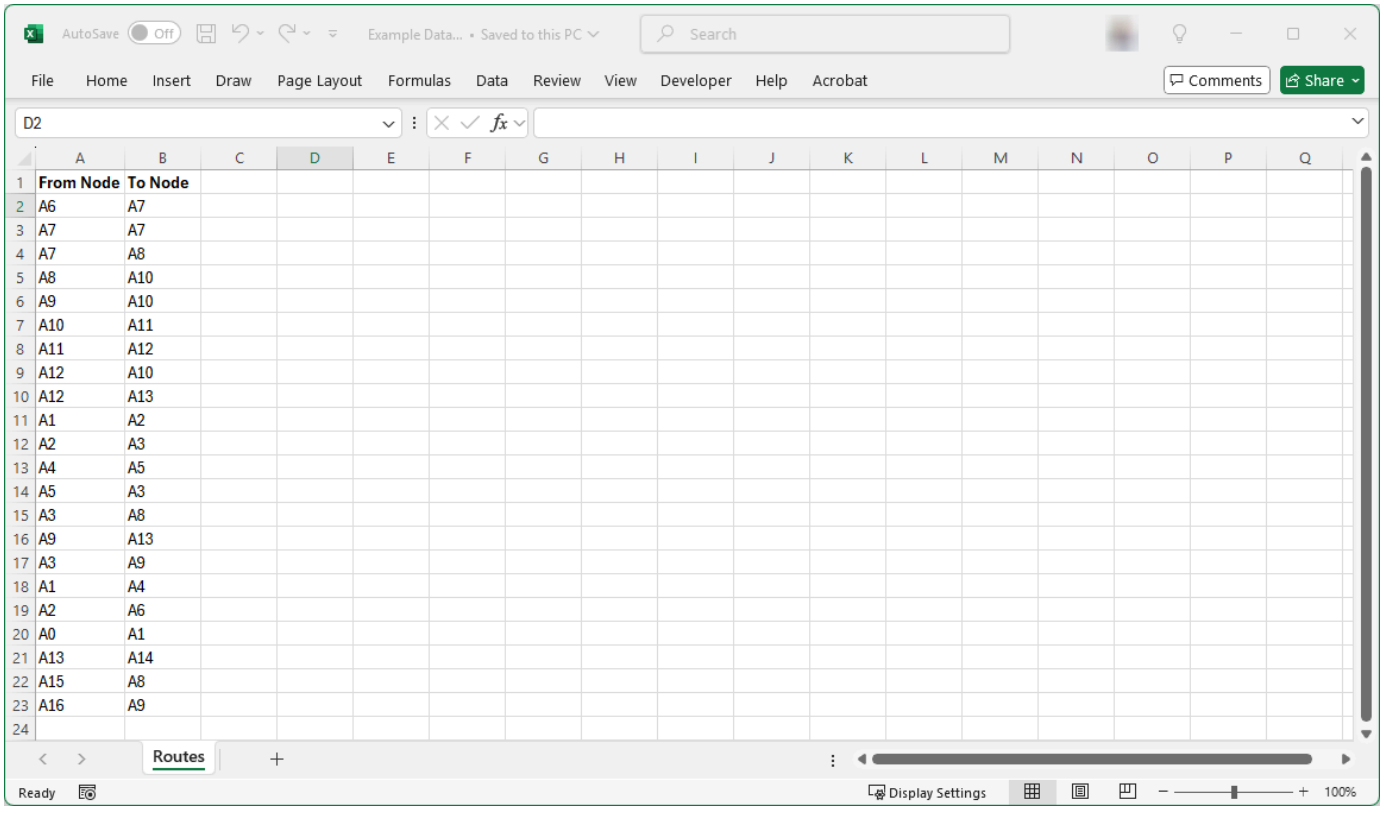
- `MAX DEPTH` - Specifies an **optional** integer value which specifies the maximum number of branches preceding, or following the node specified by `WHERE VALUE`. For example, `3 AS [MAX DEPTH]` says to constrain the search to 3 edge ranks which follow a node (Use Case

3), or 3 edge ranks which precede a node (Use Case 4), depending upon the direction of the search.

Scenario

Assume you have an Excel workbook with a worksheet named `Routes` containing 2 columns listing nodes in random order. The columns specify which node is the next node (i.e. `From Node` → `To Node`).

The data appears as:

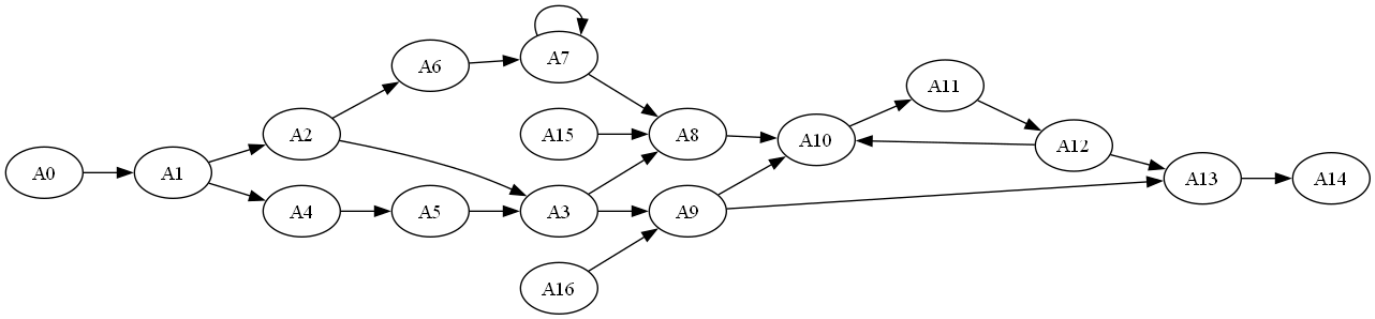


	From Node	To Node
2	A6	A7
3	A7	A7
4	A7	A8
5	A8	A10
6	A9	A10
7	A10	A11
8	A11	A12
9	A12	A10
10	A12	A13
11	A1	A2
12	A2	A3
13	A4	A5
14	A5	A3
15	A3	A8
16	A9	A13
17	A3	A9
18	A1	A4
19	A2	A6
20	A0	A1
21	A13	A14
22	A15	A8
23	A16	A9
24		

Use Case 1 - Graph Everything

The simplest way to graph this data is to use the following SQL query:

```sql
SELECT [From Node] AS [ITEM], [To Node] AS [RELATED ITEM] FROM [Routes$]
```

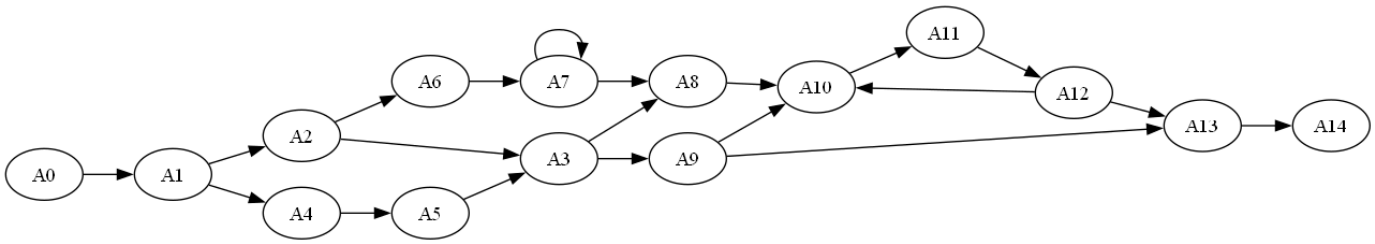


The issue with this query is that it only displays the entire graph and cannot be used to extract a subset tree. Lets resolve the issue by using a Tree Query.

Use Case 2 - Forward Search from a Node

This recursive query specifies a **START** node. It will show all the branches which flow from node `A0`.

```sql
SELECT
  'SELECT [To Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM]
     FROM [Routes$] WHERE [From Node] = ''{WHERE VALUE}'''
              AS [TREE QUERY],
  'A0'        AS [WHERE VALUE],
  'To Node'   AS [WHERE COLUMN]
```

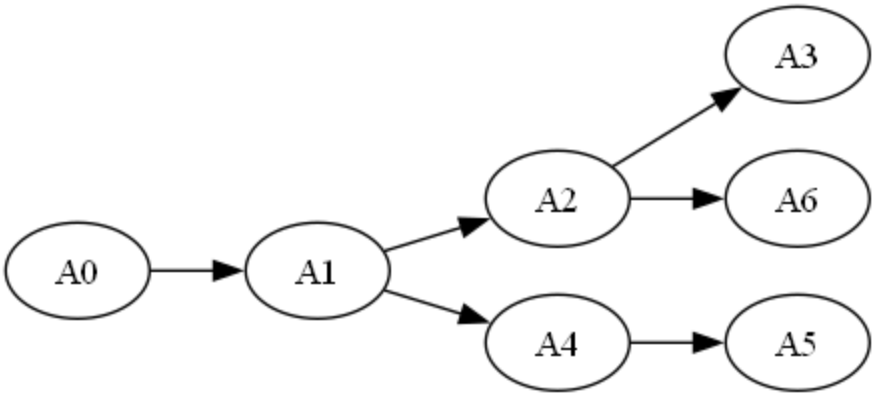


Notice that this graph looks very similar to the graph in Use Case 1, however nodes `A15` and `A16` are not present as there are no branches originating from node `A0` which connect to `A15` or `A16`.

Use Case 3 - Forward Search from a Node, with Max Depth

This query specifies a **START** node. It will show **up to 3 levels** of branches which flow from node `A0` (i.e. root node + 3 branches).

```sql
SELECT
  'SELECT [To Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM]
     FROM [Routes$] WHERE [From Node] = ''{WHERE VALUE}'''
              AS [TREE QUERY],
  'A0'        AS [WHERE VALUE],
  'To Node'   AS [WHERE COLUMN],
  3           AS [MAX DEPTH]
```



Notice that the tree stops after 3 levels of branches originating from node `A0` have been recursed.

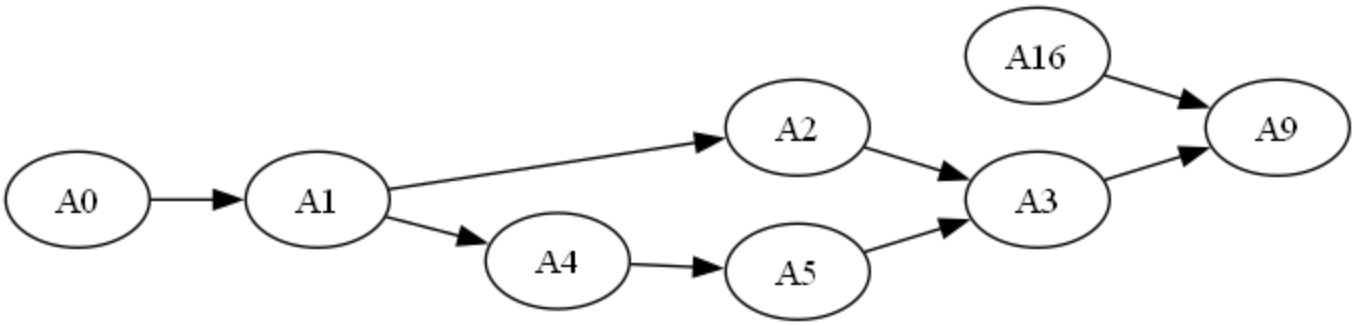
Use Case 4 - Backward Search from a Node

This query specifies an **END** node. It will show **all** the branches which flow to node `A9`.

Notice that this query is changed from Use Case 3 to use `[To Node]` in the base case portion of the query `WHERE [From Node] = ''{WHERE VALUE}''` and set `From Node` as the `WHERE COLUMN` as the recursive member.

```sql
SELECT
  'SELECT [From Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM]
     FROM [Routes$] WHERE [To Node] = ''{WHERE VALUE}'''
              AS [TREE QUERY],
```

```
    'A9'        AS [WHERE VALUE],
    'From Node' AS [WHERE COLUMN]
```



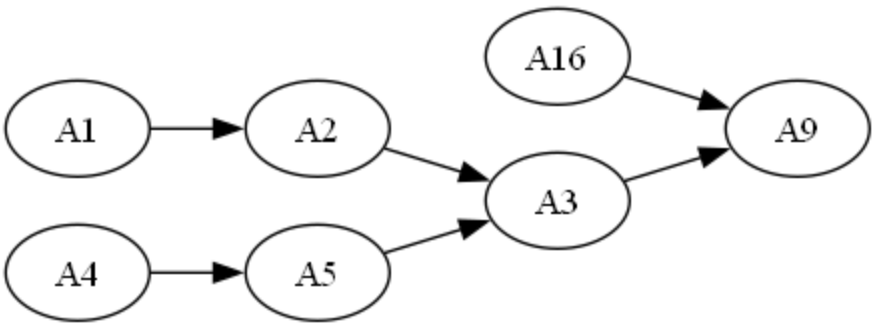
Notice in this Use Case the graph stops when node `A9` is reached. In this example node `A16` is present because it connects directly to node `A9`. Nodes `A6`, `A7`, and `A8` are not present as those routes never connect to node `A9`.

Use Case 5 - Backward Search from a Node, with Max Depth

This query specifies an **END** node, and the `[MAX DEPTH]` parameter. It will show **up to 3 levels** of branches which precede and flow to node `A9` (i.e. 3 branches + final node).

sql

```
SELECT
  'SELECT [From Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM]
     FROM [Routes$] WHERE [To Node] = ''{WHERE VALUE}'''
              AS [TREE QUERY],
  'A9'        AS [WHERE VALUE],
  'From Node' AS [WHERE COLUMN],
  3           AS [MAX DEPTH]
```



Notice in this Use Case that 3 levels of branches preceding node `A9` are present. The connection between node `A1` and `A4` is not present because the `MAX DEPTH` limit was reached with node `A1`, therefore `A1` was not recursed to determine its connections.

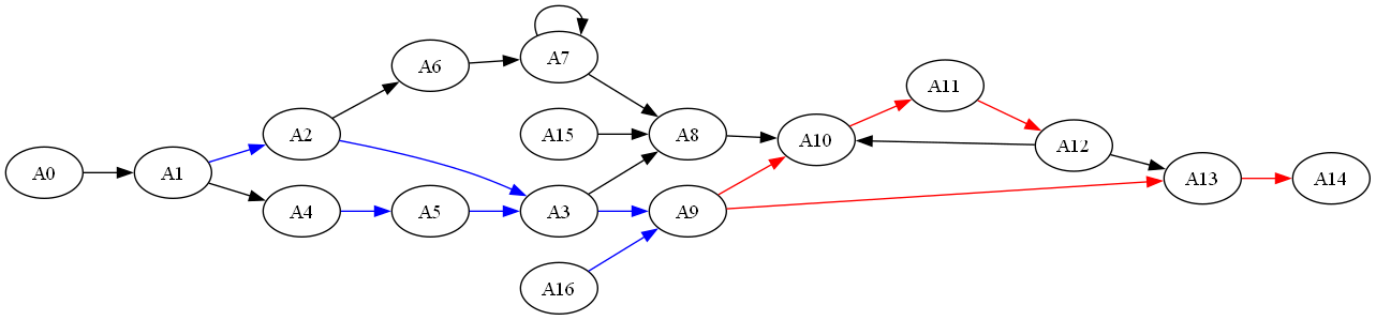
Use Case 6 - Combining Queries to Highlight Edges

This Use Case shows how you can highlight edges in a graph by recursing nodes and applying style attributes. It combines the examples above to show the entire graph. It then shows 3 levels of routes leading to node `A9` in blue, and 3 levels of routes departing node `A9` in red.

Note: the `strict` edge option on the `Graphviz` ribbon tab is enabled to prevent duplicate edges from being drawn.

```sql
-- Step 1 - Get all the routes

SELECT [From Node] AS [ITEM], [To Node] AS [RELATED ITEM] FROM [Routes$]

-- Step 2 - Recurse 3 branches of nodes which converge on node A9.
-- Change the edge color to blue

SELECT
  'SELECT [From Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM],
     ''color=blue'' as [Attributes]
     FROM [Routes$] WHERE [To Node] = ''{WHERE VALUE}'''
              AS [TREE QUERY],
  'A9'        AS [WHERE VALUE],
  'From Node' AS [WHERE COLUMN],
  3           AS [MAX DEPTH]

-- Step 3 - Recurse 3 branches of nodes which branch out from node A9.
-- Change the edge color to red

SELECT
  'SELECT [To Node], [From Node] AS [ITEM], [To Node] AS [RELATED ITEM],
     ''color=red'' AS [Attributes]
```

```
    FROM [Routes$] WHERE [From Node] = ''{WHERE VALUE}'''
            AS [TREE QUERY],
'A9'        AS [WHERE VALUE],
'To Node'   AS [WHERE COLUMN],
3           AS [MAX DEPTH]
```



Try it Yourself

The files used in these examples are contained in the `\Relationship Visualizer\samples\12 - Using SQL - Trees` directory in the zip file download.

Iterate SQL Results

Iterative SQL queries allow a query to use a result value produced by a previous parent query.

Iterate vs. Enumerate?

Within Relationship Visualizer, a distinction is made between **iteration** and **enumeration**:

- **Iterate** refers to stepping through each item in a result set one by one — without tracking position or count. Each item becomes input for a follow-up SQL query.
- **Enumerate** refers to iteration **with explicit tracking of position, index, or count**, using a counter that increments from a starting value to a stopping value. Enumeration is used when the logic depends on the numeric sequence itself. For example, "generate rows from 1 to 12," "step through values by 5," or "fill in missing years."

Scenario

You want to build a timeline graph showing the evolution of Unix shells. Assume you have an Excel workbook with a worksheet named `shells` that contains two columns: one for the year and one for a piece of information associated with that year. In this example, each year is linked to the names of Unix shells introduced during that time. The goal is to process the data so that all shells from the same year appear on the same rank in the final visualization.

The data appears as:

Year	Shell
future	ksh-POSIX
future	POSIX
1972	Thompson

Year	Shell
1976	Bourne
1976	Mashey
1978	Formshell
1978	csh
1980	esh
1980	vsh
1982	ksh
1982	System-V
1984	v9sh
1984	tcsh
1986	ksh-i
1988	rc
1988	KornShell
1988	Perl
1990	Bash
1990	tcl

Step By Step Guide

Step 1 – Place all the values for a given year in a subgroup

Relationship Visualizer includes a SQL extension that lets you define subgroups directly within a query. This is useful when you want multiple items to share the same position or

rank in the final diagram. In this scenario, each year may have several shells associated with it, and we want all shells from the same year to appear together.

You can create these subgroups using the `CREATE RANK` clause, shown below:

```SQL
TRUE AS [CREATE RANK]
```

If we want to place all the shells from 1988 on the same rank, we would write the query as follows:

```SQL
SELECT [Shell] AS [ITEM],
       TRUE    AS [CREATE RANK],
       'same'  AS [RANK]
FROM  [shells$]
WHERE [YEAR] = '1988'
```

This will insert the following subgroup into the DOT source:

```
{ rank="same"; "1988"; "KornShell"; "Perl"; "rc"; }
```

Step 2 – Get a list of unique years

The previous approach works for a single year, but we want to repeat the process for every year in the dataset without hard-coding values into the SQL. The next step is to retrieve a list of all years, removing duplicates by using the standard `SELECT DISTINCT` command:

```SQL
SELECT DISTINCT [Year] AS [ID] FROM [shells$]
```

ID
future

ID
1972
1976
1978
1980
1982
1984
1986
1988
1990

Step 3 – Iterate the unique years, generating a subgroup for each year

This step combines Step 2 with Step 1. For this step, we supply an additional SQL extension flag introduced in Version 10.

```SQL
TRUE AS [ITERATE]
```

This flag tells Relationship Visualizer that the query is a special iterative query. It instructs the engine to run two SQL statements: the first generates the list of IDs, and the second is executed once for each ID in that list.

You pass the queries using the column names

- `[SQL FOR ID]`
- `[SQL FOR DATA]`

The `[SQL FOR ID]` query producing the ID values is:

```sql
SELECT DISTINCT [Year] AS [ID] FROM [shells$]
```

The `[SQL FOR DATA]` query to iterate the ID values is:

```sql
SELECT [Shell] AS [ITEM], TRUE AS [CREATE RANK], ''same'' AS [RANK] FROM [shells$] WHE
```

Combining these into a single SQL statement looks like this. Note that because the SQL statements are being passed as strings, any string values inside the query—such as `'{ID}'`—must be escaped with doubled single quotes, like `''{ID}''`.

```SQL
SELECT
  'SELECT DISTINCT [Year] AS [ID] FROM [shells$]' AS [SQL FOR ID],
  'SELECT [Shell] AS [ITEM], TRUE AS [CREATE RANK], ''same'' AS [RANK] FROM [shells$] W
  TRUE AS [ITERATE]
```

Relationship Visualizer will loop through the ID list produced by the first query. Each ID value is substituted into the second query wherever the `{ID}` placeholder appears. The modified query is then executed, and the results are written to the `data` worksheet.

Using the sample data above, this pair of SQL statements adds the following entries to the `data` worksheet:

```
{ rank="same"; "Thompson"; }
{ rank="same"; "Bourne"; "Mashey"; }
{ rank="same"; "csh"; "Formshell"; }
{ rank="same"; "esh"; "vsh"; }
{ rank="same"; "ksh"; "System-V"; }
{ rank="same"; "tcsh"; "v9sh"; }
{ rank="same"; "ksh-i"; }
{ rank="same"; "KornShell"; "Perl"; "rc"; }
{ rank="same"; "Bash"; "tcl"; }
{ rank="same"; "ksh-POSIX"; "POSIX"; }
```

Step 4 – Add the year to the subgroups

The subgroups based on year have been created, but notice that the year itself is not included. How do we know which shells belong to which years?

In SQL, the `UNION` operator functions like an "AND" that allows you to run additional queries and return the unique combined results.

The following statement can supply the year:

```sql
SELECT [Year] AS [ITEM] WHERE [YEAR] = '{ID}'
```

`UNION` has a restriction that all SQL statements must return the same list of fields. To comply with this rule, the statement becomes:

```sql
SELECT [Year]  AS [ITEM], TRUE AS [CREATE RANK], 'same' AS [RANK] FROM [shells$] WHERE
```

The `UNION` is added to the statement as follows:

```sql
SELECT
  'SELECT DISTINCT [Year] AS [ID] FROM [shells$]' AS [SQL FOR ID],
  'SELECT [Year]  AS [ITEM], TRUE AS [CREATE RANK], ''same'' AS [RANK] FROM [shells$] W
     UNION
   SELECT [Shell] AS [ITEM], TRUE AS [CREATE RANK], ''same'' AS [RANK] FROM [shells$] W
  TRUE AS [ITERATE]
```

When the query is run, the following subgroups are produced:

```
{ rank="same"; "1972"; "Thompson"; }
{ rank="same"; "1976"; "Bourne"; "Mashey"; }
{ rank="same"; "1978"; "csh"; "Formshell"; }
{ rank="same"; "1980"; "esh"; "vsh"; }
{ rank="same"; "1982"; "ksh"; "System-V"; }
{ rank="same"; "1984"; "tcsh"; "v9sh"; }
```

```
  { rank="same"; "1986"; "ksh-i"; }
  { rank="same"; "1988"; "KornShell"; "Perl"; "rc"; }
  { rank="same"; "1990"; "Bash"; "tcl"; }
  { rank="same"; "future"; "ksh-POSIX"; "POSIX"; }
```

Now the subgroups contain both the year and the shells introduced during that year.

Step 5 – Build a timeline

Next, we want to build a timeline. For this, we use another Relationship Visualizer SQL extension.

Similar to the `CREATE RANK` keyword, there is a `CREATE EDGES` keyword, specified as:

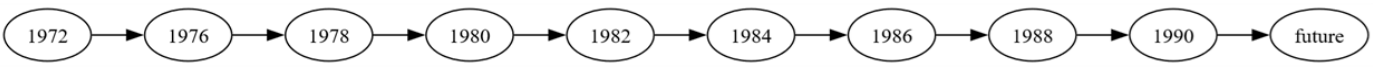
```sql
TRUE AS [CREATE EDGES]
```

When this query runs, it generates a set of rows in the `data` worksheet that create an edge from each item to the next. If the data contains `a`, `b`, `c`, the resulting edges will be `a` → `b` → `c`.

The single statement:

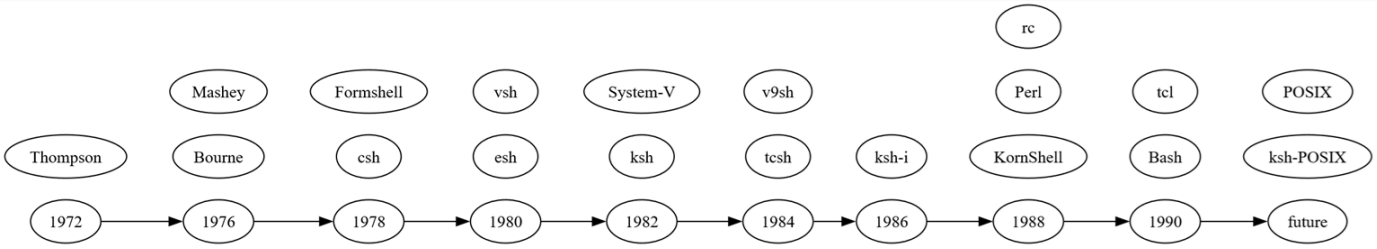
```sql
SELECT DISTINCT [Year] AS [ITEM],
       TRUE            AS [CREATE EDGES]
FROM   [Shells$]
WHERE  [Year] IS NOT NULL
ORDER BY [Year] ASC
```

produces the following chain of edges from year to year.



Step 6 – Connect the subgroups to the timeline

This step combines the SQL statements from Step 5 and Step 6. The resulting graph becomes:



Step 7 – Depict relationships between items

Our data also includes a worksheet named `evolution`, which tells us which shells preceded other shells. The data looks as follows:

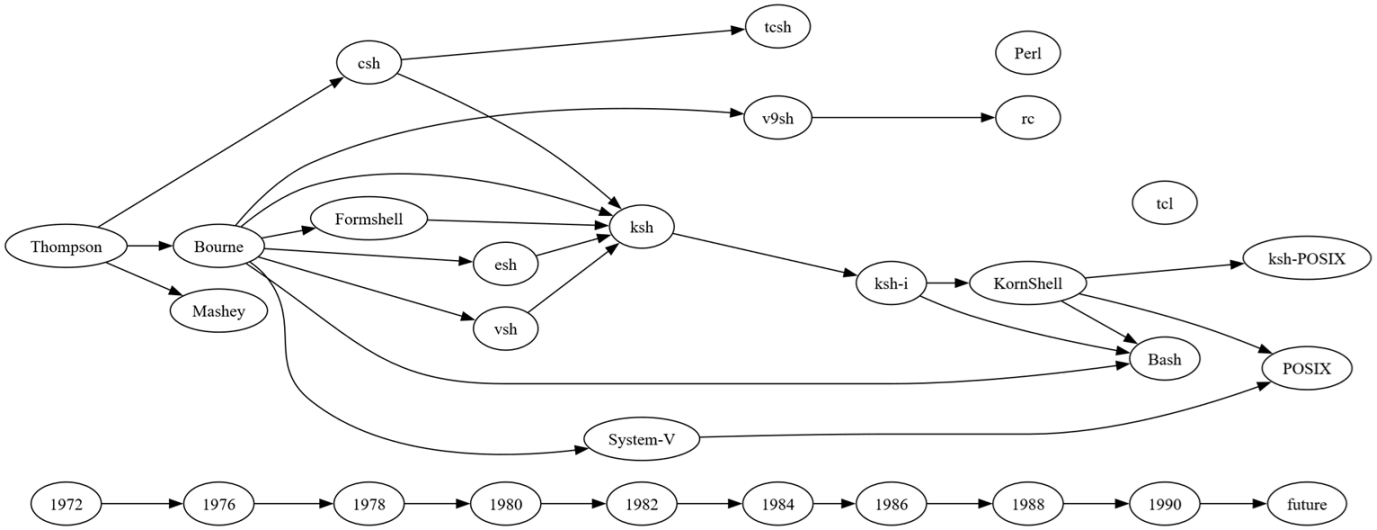
From Shell	To Shell
Thompson	Bourne
Thompson	Mashey
Thompson	csh
Bourne	v9sh
Bourne	ksh
Bourne	esh
Bourne	vsh
Bourne	Formshell
Bourne	Bash
Bourne	System-V
Formshell	ksh
csh	ksh
ksh	ksh-i

From Shell	To Shell
csh	tcsh
System-V	POSIX
v9sh	rc
KornShell	ksh-POSIX
KornShell	POSIX
KornShell	Bash
esh	ksh
vsh	ksh
ksh-i	KornShell
ksh-i	Bash

It is a simple change to add the following SQL statement:

```sql
SELECT [From Shell] AS [ITEM],
       [To Shell]   AS [RELATED ITEM]
FROM   [evolution$]
```

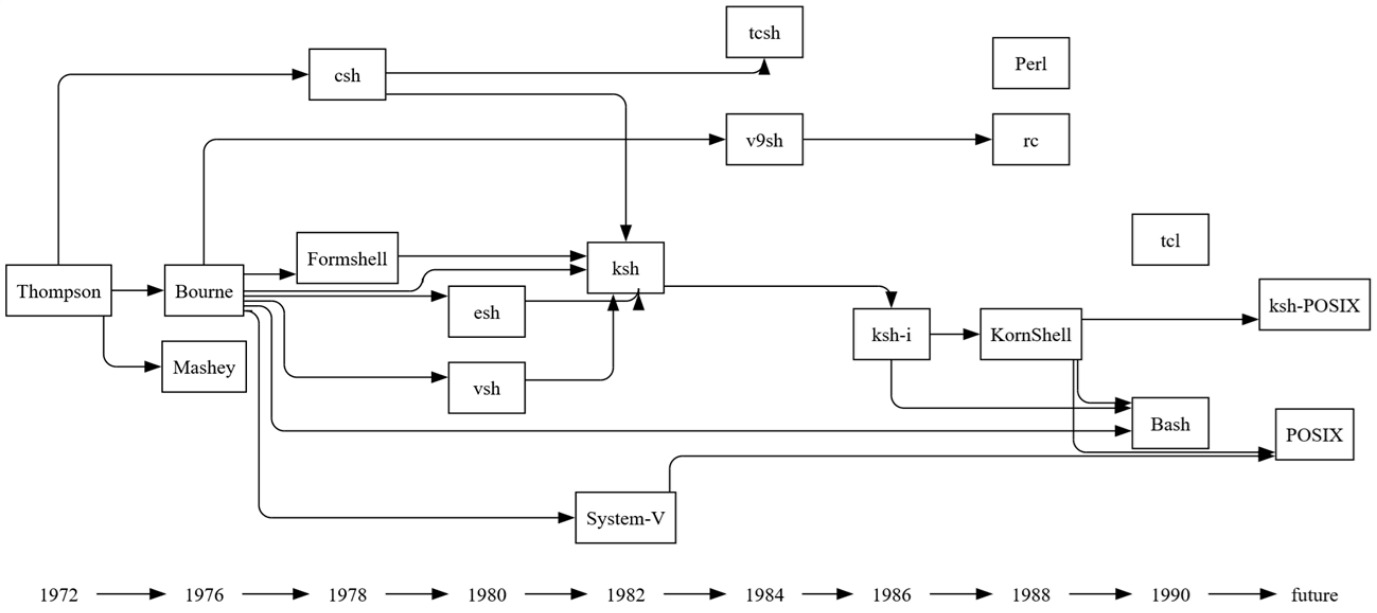
which changes the graph to:



Step 8 – Add style and adjust layout

The last step is to create styles and apply them to the nodes and edges. This is standard Relationship Visualizer work, so it won't be detailed here. Refer to the sample spreadsheet if you'd like to explore the styling options.

The fully styled graph becomes:



Try it Yourself

This example is included in the samples in the Relationship Visualizer zip file in the directory `16 - Using SQL - Iteration`.

Creating Organization Charts

A frequent question is: *"Can you create organization charts using the Relationship Visualizer?"*
The answer is **YES!**

The tool's support for recursive SQL queries and parent–child relationships makes it well-suited for generating multi-level organizational hierarchies. As long as your dataset includes clear parent and child identifiers, the Relationship Visualizer can automatically construct a complete org chart—whether it's a simple two-level structure or a deeply nested hierarchy.

What is Recursion?

Recursion works by having a function repeatedly call itself, each time handling a smaller piece of the problem. The process continues until it reaches a base case that tells it when to stop.

Recursive SQL Queries

Relationship Visualizer Version 7.2 introduced support for recursive SQL queries, enabling the automatic construction of hierarchical structures such as organization charts, dependency trees, and connected data paths. Recursive queries allow the tool to walk parent–child relationships repeatedly, building multi-level hierarchies from simple row-based data.

To use this feature effectively, the candidate dataset must include the key columns that define each relationship:

- A **parent** node
- A **child** node

These columns form the backbone of the hierarchy, allowing the recursive query engine to traverse upward or downward through the structure and generate a complete, layered

graph.

A recursive query consists of two essential parts:

1. **Anchor the Base Case**
   This defines the starting point for recursion—typically the top-level parent in the hierarchy (for example, the CEO in an organization chart or the root node in a dependency tree). The anchor member returns the initial set of rows from which the hierarchy will begin.

2. **Define the Recursive Member**
   This portion of the query describes how to repeatedly join the growing result set back to the source table. Each iteration locates the next level of child nodes, allowing the hierarchy to expand downward (or upward) until no additional relationships remain.

SQL Extensions

Four new keywords unique to Relationship Visualizer have been defined to enable recursive queries:

- `TREE QUERY` - Passed as a literal string, the presence of `TREE QUERY` in the SQL result set tells the SQL engine to execute the associated SQL recursively.

  The SQL **must** contain the symbolic value `''{WHERE VALUE}''` which will be replaced as each node is traversed with the value of the current node.

  Note that since `TREE QUERY` is passed as a literal string, any strings in the SQL which would normally be specified with `'` delimiters must be escaped as `''`.

- `WHERE VALUE` provides the value which anchors the base case. For example, in an organization chart it could be the ID of the CEO. For a subway line, it could be the starting subway station.

- `WHERE COLUMN` - Specifies the name of the column which defines the recursive member. It is the name of the column containing the value which should be searched next. For

example, in an organization chart it could be column name of the column containing the ID of the employee's manager.

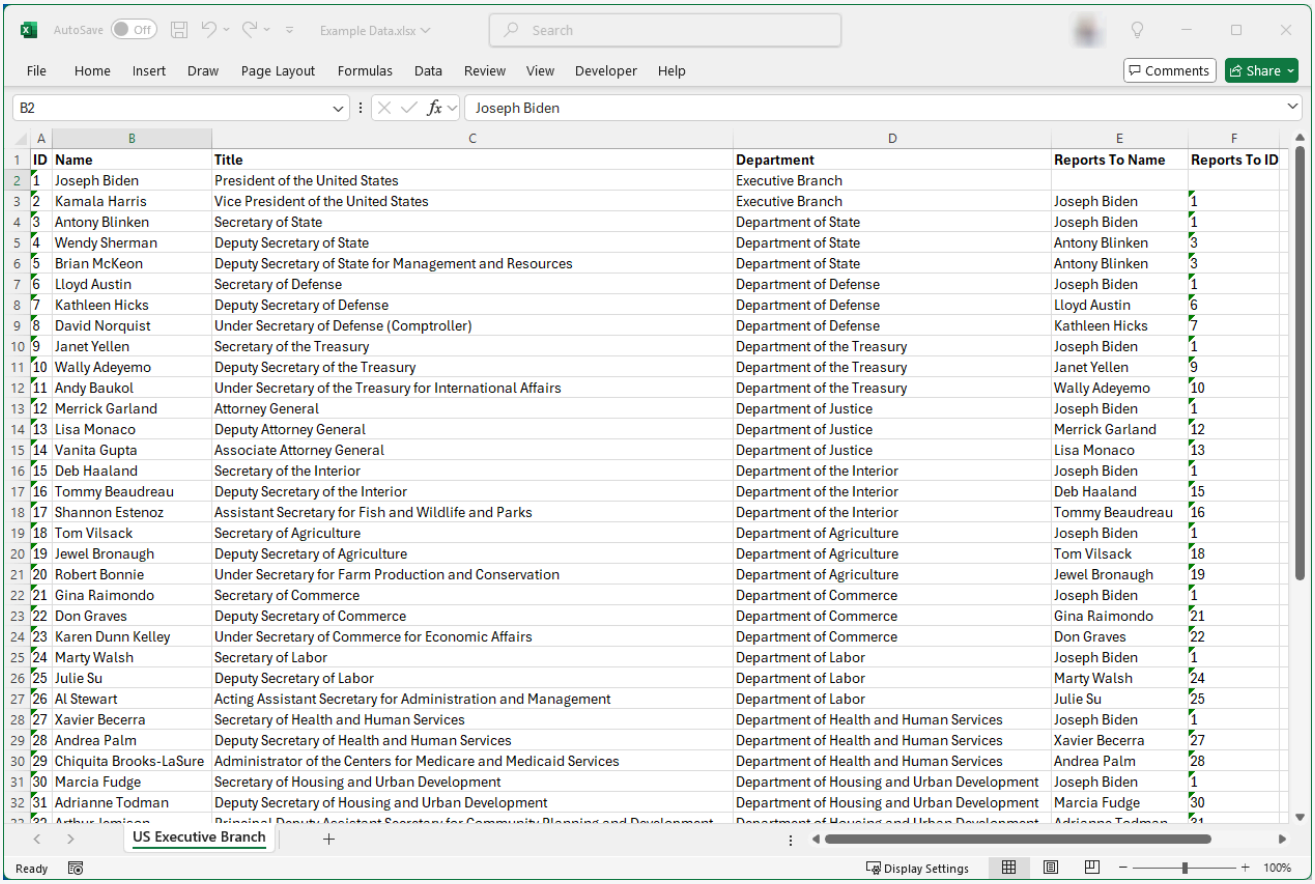
- `MAX DEPTH` - Specifies an **optional** integer value which specifies the maximum number of branches preceding, or following the node specified by `WHERE VALUE`. For example, `3 AS [MAX DEPTH]` says to constrain the search to 3 edge ranks which follow a node , or 3 edge ranks which precede a node , depending upon the direction of the search.

Scenario

You need to understand how the Executive Branch of the United States government is organized, so you want to create an organization chart.

> **No political messages are intended**
>
> A sample dataset of historical data is used purely for demonstration. This data was collected using AI and reflects the Executive Branch in place at the time the data was collected.

Assume you have an Excel workbook with a worksheet named `US Executive Branch` containing columns such as **Name**, **Title**, **Department**, and **Reports To**. The data appears as follows:



ID	Name	Title	Department	Reports To Name	Reports To ID
1	Joseph Biden	President of the United States	Executive Branch		
2	Kamala Harris	Vice President of the United States	Executive Branch	Joseph Biden	1
3	Antony Blinken	Secretary of State	Department of State	Joseph Biden	1
4	Wendy Sherman	Deputy Secretary of State	Department of State	Antony Blinken	3
5	Brian McKeon	Deputy Secretary of State for Management and Resources	Department of State	Antony Blinken	3
6	Lloyd Austin	Secretary of Defense	Department of Defense	Joseph Biden	1
7	Kathleen Hicks	Deputy Secretary of Defense	Department of Defense	Lloyd Austin	6
8	David Norquist	Under Secretary of Defense (Comptroller)	Department of Defense	Kathleen Hicks	7
9	Janet Yellen	Secretary of the Treasury	Department of the Treasury	Joseph Biden	1
10	Wally Adeyemo	Deputy Secretary of the Treasury	Department of the Treasury	Janet Yellen	9
11	Andy Baukol	Under Secretary of the Treasury for International Affairs	Department of the Treasury	Wally Adeyemo	10
12	Merrick Garland	Attorney General	Department of Justice	Joseph Biden	1
13	Lisa Monaco	Deputy Attorney General	Department of Justice	Merrick Garland	12
14	Vanita Gupta	Associate Attorney General	Department of Justice	Lisa Monaco	13
15	Deb Haaland	Secretary of the Interior	Department of the Interior	Joseph Biden	1
16	Tommy Beaudreau	Deputy Secretary of the Interior	Department of the Interior	Deb Haaland	15
17	Shannon Estenoz	Assistant Secretary for Fish and Wildlife and Parks	Department of the Interior	Tommy Beaudreau	16
18	Tom Vilsack	Secretary of Agriculture	Department of Agriculture	Joseph Biden	1
19	Jewel Bronaugh	Deputy Secretary of Agriculture	Department of Agriculture	Tom Vilsack	18
20	Robert Bonnie	Under Secretary for Farm Production and Conservation	Department of Agriculture	Jewel Bronaugh	19
21	Gina Raimondo	Secretary of Commerce	Department of Commerce	Joseph Biden	1
22	Don Graves	Deputy Secretary of Commerce	Department of Commerce	Gina Raimondo	21
23	Karen Dunn Kelley	Under Secretary of Commerce for Economic Affairs	Department of Commerce	Don Graves	22
24	Marty Walsh	Secretary of Labor	Department of Labor	Joseph Biden	1
25	Julie Su	Deputy Secretary of Labor	Department of Labor	Marty Walsh	24
26	Al Stewart	Acting Assistant Secretary for Administration and Management	Department of Labor	Julie Su	25
27	Xavier Becerra	Secretary of Health and Human Services	Department of Health and Human Services	Joseph Biden	1
28	Andrea Palm	Deputy Secretary of Health and Human Services	Department of Health and Human Services	Xavier Becerra	27
29	Chiquita Brooks-LaSure	Administrator of the Centers for Medicare and Medicaid Services	Department of Health and Human Services	Andrea Palm	28
30	Marcia Fudge	Secretary of Housing and Urban Development	Department of Housing and Urban Development	Joseph Biden	1
31	Adrianne Todman	Deputy Secretary of Housing and Urban Development	Department of Housing and Urban Development	Marcia Fudge	30

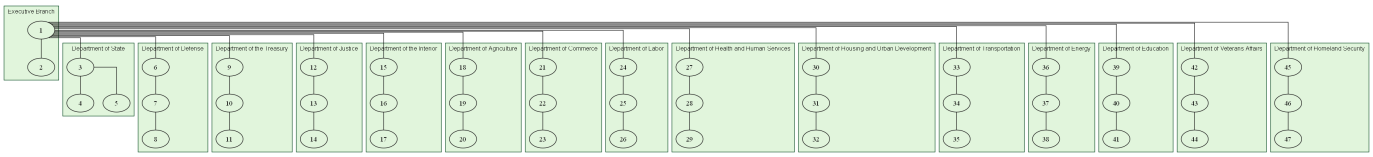
Standard Query

The traditional way to graph this data was to use a query such as:

```sql
SELECT [Reports To ID] AS [ITEM],    [ID]        AS [RELATED ITEM],
       [Department]    AS [CLUSTER], 'Border 2 ' AS [CLUSTER STYLE NAME]
FROM   [US Executive Branch$]
```

which relates the `Reports To ID` to the `ID`, and clusters the nodes by Department, using the `Border 2` style on the `styles` worksheet.

Running the SQL produces the following (undirected) graph:



The issue with this approach is that it graphs the entire organization. For instance, if you only want to graph the department heads, there isn't a specific column in the data to filter by.

Tree Query

Max Depth

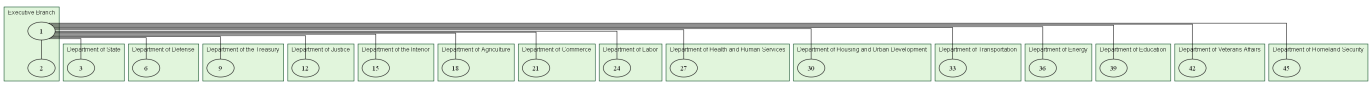
A tree query which specifies a `Max Depth` will allow us to extract just the department heads from the full organization.

We write the Tree Query as follows:

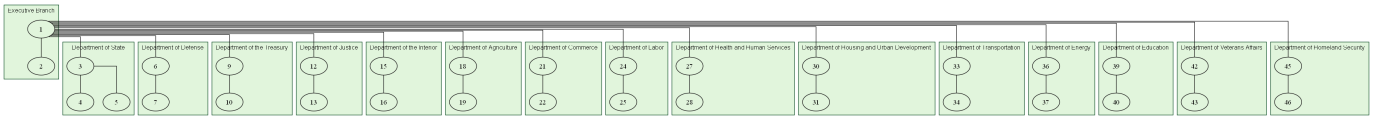
```sql
SELECT
  'SELECT [ID], [Reports To ID] AS [ITEM],    [ID] AS [RELATED ITEM],
               [Department]    AS [CLUSTER], ''Border 2 '' AS [CLUSTER STYLE NAME]
          FROM [US Executive Branch$] WHERE [Reports To ID] = ''{WHERE VALUE}'''
       AS [TREE QUERY],
  '1'  AS [WHERE VALUE],
  'ID' AS [WHERE COLUMN],
  1    AS [MAX DEPTH]
```

Where Value

In this query we are using `ID = 1` (*President of the United States*), and a `Max Depth = 1`. The graph changes to:



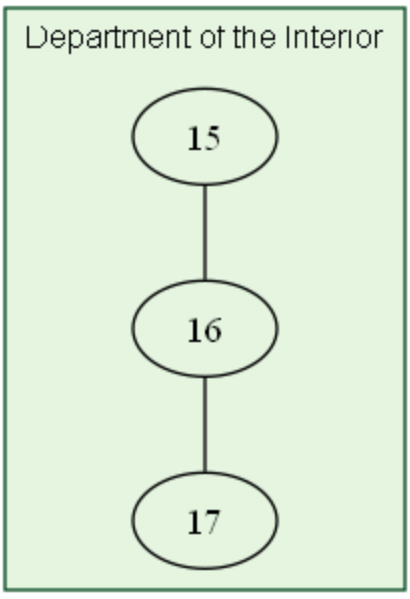
If we change the `Max Depth` to `2`, the graph changes to:



Likewise we can choose the `ID` of a department head and obtain the organization chart for a single department. `15` is the `ID` for the head of the *Department of the Interior*. If we change the `WHERE VALUE` in the query to `15`, and `MAX DEPTH` to `2`, the query becomes:

```sql
SELECT
  'SELECT [ID], [Reports To ID] AS [ITEM],    [ID] AS [RELATED ITEM],
               [Department]    AS [CLUSTER], ''Border 2 '' AS [CLUSTER STYLE NAME]
          FROM [US Executive Branch$] WHERE [Reports To ID] = ''{WHERE VALUE}'''
       AS [TREE QUERY],
  '15' AS [WHERE VALUE],
  'ID' AS [WHERE COLUMN],
  2    AS [MAX DEPTH]
```

and the graph changes to:



Styling the Nodes

Named Styles

The tree structure has been determined, now we want to create styled nodes with department detail.

The first step is to create a style definition on the `styles` worksheet. We use the `style designer` tab to create a style named `Department` which appears as follows:

Label Text Format

We would also like the name of the department head to appear in bold text, and the job title to appear as normal text on its own line, separated by a blank line. Graphviz supports HTML-like labels which can be used. The HTML-like label needs to follow the pattern

```
<<b>Name</b><BR/><BR/>Title>
```

for example:

```
<<b>Deb Haaland</b><BR/><BR/>Secretary of the Interior>
```

Label Line Splits

We are also concerned about the length of the names or titles being larger than the width of the rectangle we have defined. The Relationship Visualizer SQL interpreter has features which allow you to split text at the nearest blank character after a specified length, and a feature to allow you to specify a string at the point the the split.

In the SQL we can specify

- `'25' AS [SPLIT LENGTH]` to split each line in the label as close as possible to the 25th position
- `'<BR/>' [AS LINE ENDING]` as the line ending of each break in the string. The label is HTML-like, so we use an HTML break representation `<BR/>`.

  FYI, if we were using plain-text labels the line ending choices would be `\l` for left justied breaks, `\n` for centered breaks, or `\r` for right-justified breaks. If we were using a Graphviz `record` shape the line ending could be `|`. You have complete flexibility to insert any text at the point of a split in the text.

The label will be emitted to fit within the rectangle, as follows:

```
<<b>Deb Haaland</b><BR/><BR/>Secretary of the<BR/>Interior>
```

Writing the Node Query

Our first query traverses the tree and writes the `Item` is related to `Related Item` edge relationships. We now repeat the query to traverse the tree a second time, but we will only output information specific to the nodes.

The Node query is specified as:

```sql
SELECT
      'SELECT [ID] AS [ID],
              [ID] AS [ITEM],
              ''<<b>'' & [Name] & ''</b><br/><br/>'' & [Title] & ''>'' AS [LABEL],
              ''25''         AS [SPLIT LENGTH],
              ''<BR/>''      AS [LINE ENDING],
              ''Department'' AS [STYLE NAME]
       FROM   [US Executive Branch$]
       WHERE  [Reports To ID] = ''{WHERE VALUE}'''
                    AS [TREE QUERY],
      '15'  AS [WHERE VALUE],
      'ID' AS [WHERE COLUMN],
```

```
       2 AS [MAX DEPTH],

       [ID] AS [ITEM],
       '<<b>' & [Name] & '</b><br/><br/>' & [Title] & '>' AS [LABEL],
       '25'         AS [SPLIT LENGTH],
       '<BR/>'      AS [LINE ENDING],
       'Department' AS [STYLE NAME]
FROM  [US Executive Branch$]
WHERE [ID] = '15'
```

where the label is specified as

```sql
              ''<<b>'' & [Name] & ''</b><br/><br/>'' & [Title] & ''>'' AS [LABEL],
```

to concatenate column values with literal strings to create the HTML-like label described above.

The line splitting is specified as:

```sql
              ''25''         AS [SPLIT LENGTH],
              ''<BR/>''      AS [LINE ENDING],
```

The Node style is specified as:

```sql
              ''Department'' AS [STYLE NAME]
```

One additional thing you **must** notice is that this syntax repeats as:

```sql
       [ID] AS [ITEM],
       '<<b>' & [Name] & '</b><br/><br/>' & [Title] & '>' AS [LABEL],
       '25'         AS [SPLIT LENGTH],
       '<BR/>'      AS [LINE ENDING],
       'Department' AS [STYLE NAME]
```

The reason for this repeated syntax is related to the dual-query nature of a recursive tree query. At the beginning of this document, the concept of *Anchoring the Base Case* is discussed. This syntax outputs the node information for the base case (in this example, `ID = 25`). To create a complete organization chart, we must output the node information for the base case and for each branch of the tree as it is traversed.

Important: Pay very close attention to how literal strings are represented in the *Base Case* query, and the *Recursive Member Case* query to avoid SQL syntax errors.

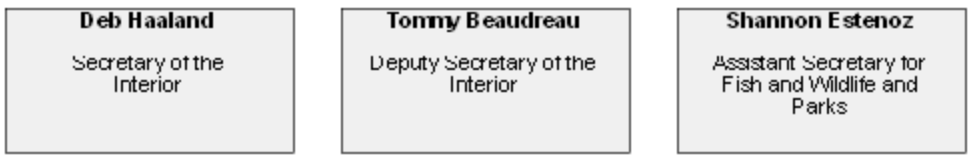
- The *Base Case* query uses 1 single quote to denote a string.

```sql
'<<b>' & [Name] & '</b><br/><br/>' & [Title] & '>' AS [LABEL],
'25'         AS [SPLIT LENGTH],
```

- The *Recursive Member* query uses 2 single quotes to denote a string within the `TREE QUERY` string (i.e. strings within a string require 2 single quotes).

```sql
''<<b>'' & [Name] & ''</b><br/><br/>'' & [Title] & ''>'' AS [LABEL],
''25''         AS [SPLIT LENGTH],
```

When we run the query, the nodes are output as:



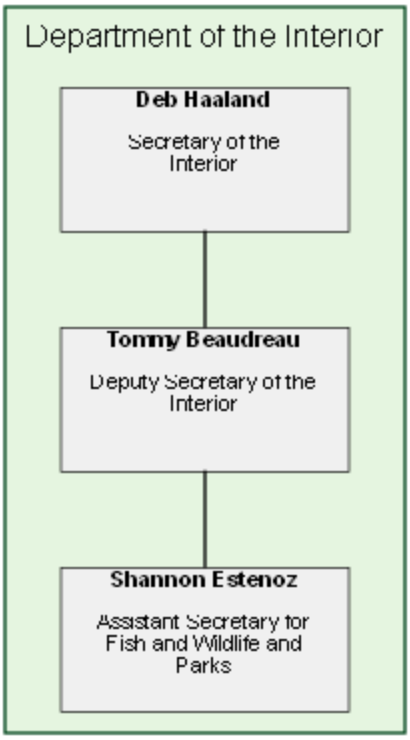
Combining the Edges and Nodes

Now it just a matter of running both SQL statements to combine the edges and nodes.

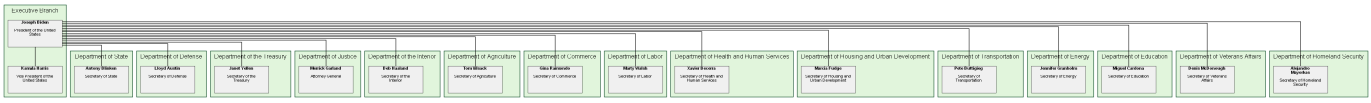
```sql
SELECT
  'SELECT [ID], [Reports To ID] AS [ITEM],    [ID] AS [RELATED ITEM],
              [Department]    AS [CLUSTER], ''Border 2 '' AS [CLUSTER STYLE NAME]
         FROM [US Executive Branch$] WHERE [Reports To ID] = ''{WHERE VALUE}'''
```

```
       AS [TREE QUERY],
  '15' AS [WHERE VALUE],
  'ID' AS [WHERE COLUMN],
  2    AS [MAX DEPTH]

SELECT
      'SELECT [ID] AS [ID],
              [ID] AS [ITEM],
              ''<<b>'' & [Name] & ''</b><br/><br/>'' & [Title] & ''>'' AS [LABEL],
              ''25''         AS [SPLIT LENGTH],
              ''<BR/>''      AS [LINE ENDING],
              ''Department'' AS [STYLE NAME]
       FROM   [US Executive Branch$]
       WHERE  [Reports To ID] = ''{WHERE VALUE}'''
                    AS [TREE QUERY],
      '15'  AS [WHERE VALUE],
      'ID' AS [WHERE COLUMN],
      2 AS [MAX DEPTH],

      [ID] AS [ITEM],
      '<<b>' & [Name] & '</b><br/><br/>' & [Title] & '>' AS [LABEL],
      '25'         AS [SPLIT LENGTH],
      '<BR/>'      AS [LINE ENDING],
      'Department' AS [STYLE NAME]
FROM  [US Executive Branch$]
WHERE [ID] = '15'
```

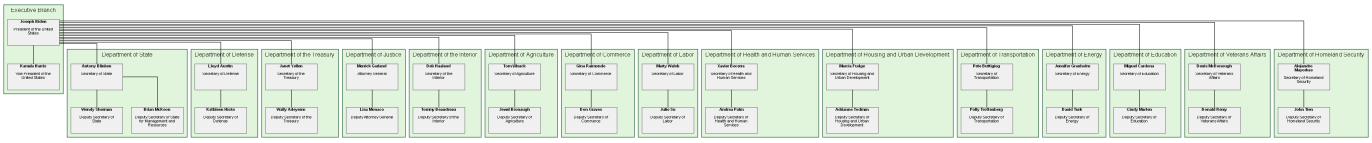
The organization chart for the *Department of the Interior* `ID=15` appears as:



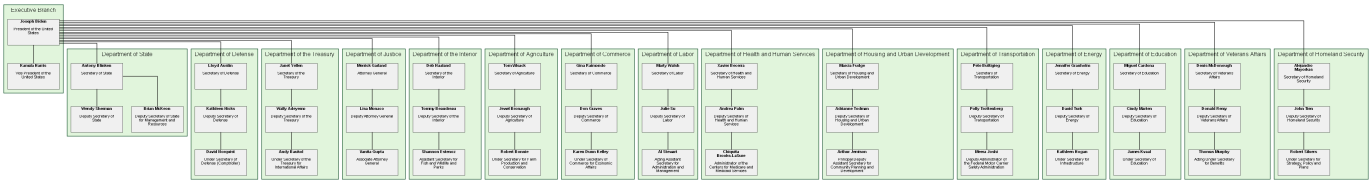
Running the queries starting with the *President of the United States* (`ID=1`) and `Max Depth = 1` shows the President's cabinet.



With `Max Depth = 2` we see the Secretaries and Deputy Secretaries.



Specifying `Max Depth = 0` removes the depth limit, and we get the full organization chart.



Sample Content

The files used in these examples are contained in the `\Relationship Visualizer\samples\13 - Using SQL - Organization Charts` directory in the zip file download.

Combining Iteration and Enumeration to Build Roadmap Timelines

Previous examples focused on iteration (processing what already exists in the data) and enumeration (sequentially generating what should exist). A roadmap requires both. Real-world timelines rarely contain perfectly aligned events: some years have many activities, others have none, and events may span multiple quarters or depend on one another. To visualize this cleanly, Relationship Visualizer must combine synthetic structure with data-driven detail.

This example demonstrates how to build a complete roadmap by layering enumerated year scaffolding, iterative subgrouping, event-to-event edges, and dependency relationships into a single coherent diagram.

Timeline Data

Timeline events and dependencies are tracked in two worksheets. The data is as follows:

"timeline" Worksheet

This timeline is composed of a set of 20 hypothetical events which are tracked to a year and quarter where they begin or end.

EventID	EventName	StartYear	StartQuarter	EndYear	EndQuarter	Stat
E001	Atlas Core R&D	2023	Q1	2025	Q4	Compl
E002	Atlas 1.0 Release	2025	Q3	2025	Q3	Compl
E003	Atlas 1.x Support	2025	Q4	2027	Q2	Or Sche

EventID	EventName	StartYear	StartQuarter	EndYear	EndQuarter	Stat
E004	Atlas 2.0 Development	2026	Q1	2027	Q4	On Sche
E005	Atlas 2.0 Release	2028	Q1	2028	Q1	Plann
E006	Beacon UI Prototype	2024	Q2	2024	Q4	Compl
E007	Beacon 1.0 Development	2025	Q1	2026	Q2	On Sche
E008	Beacon 1.0 Release	2026	Q3	2026	Q3	Plann
E009	Beacon 1.x Enhancements	2026	Q4	2027	Q4	Plann
E010	Cipher Engine Research	2022	Q3	2024	Q1	Compl
E011	Cipher Engine 1.0 Release	2024	Q2	2024	Q2	Compl
E012	Cipher 1.x Optimization	2024	Q3	2025	Q4	Compl
E013	DeltaCloud Integration	2025	Q2	2026	Q4	Behi Sche
E014	DeltaCloud 2.0 Upgrade	2027	Q1	2028	Q2	Plann
E015	Echo Services Pilot	2026	Q1	2026	Q4	On Sche
E016	Echo 1.0 Release	2027	Q1	2027	Q1	Plann

EventID	EventName	StartYear	StartQuarter	EndYear	EndQuarter	Stat
E017	Echo 1.x Support	2027	Q2	2028	Q4	Plann
E018	Fusion Platform R&D	2027	Q3	2029	Q2	Plann
E019	Fusion 1.0 Release	2029	Q3	2029	Q3	Plann
E020	Fusion 1.x Patch Cycle	2029	Q4	2030	Q2	Plann

"dependencies" Worksheet

In a small number of cases the start of an event is dependent upon the finish of another event.

Must Complete	Before the Start Of
E004	E009
E004	E005
E016	E003

Step-By-Step

Step 1 — Title the Roadmap

The roadmap begins by giving the graph a title derived from the earliest and latest years in the dataset. This ensures the diagram is self-describing and automatically adapts as new events are added. The title is styled separately so it appears as a page-level heading in the final visualization.

```sql
SELECT 'graph'                                                          AS [Item],
       CStr(MIN([StartYear])) & ' - ' & CStr(MAX([StartYear])) & ' Roadmap' AS [Label],
       'Page Title'                                                     AS [Style N
FROM   [timeline$]
```

Which appears as:

2022 - 2029 Roadmap

Step 2 — Create a Continuous Year Backbone

A roadmap must show every year in the planning horizon, not just the years that contain events. **Enumeration** is used to generate a complete numeric sequence from the minimum start year to the maximum end year. This produces a clean chain of year-to-year edges, filling in any gaps where no events occur.

```sql
SELECT TRUE AS [ENUMERATE], MIN(CLng([StartYear])) AS [START AT], MAX(CLng([EndYear]))
       '{step}' AS [Item], 'Transparent Edge' AS [Style Name],
       TRUE AS [CREATE EDGES]
FROM   [timeline$]
WHERE  IsNumeric([StartYear])
```

These edges are styled with a **transparent** appearance so they provide structure without overwhelming the event-level details.

Step 3 — Style the Year Nodes

Once the year nodes have been generated, a second **enumerated** query applies a consistent style to each one.

```sql
SELECT TRUE AS [ENUMERATE], MIN(CLng([StartYear])) AS [START AT], MAX(CLng([EndYear]))
       '{step}' AS [Item],
       '{step}' AS [Label],
       'Year'   AS [Style Name]
FROM   [timeline$]
WHERE  IsNumeric([StartYear])
```

This separates the visual identity of the timeline backbone from the event nodes that will be added later. Labels are applied directly from the enumerated step value, ensuring each year is clearly marked.

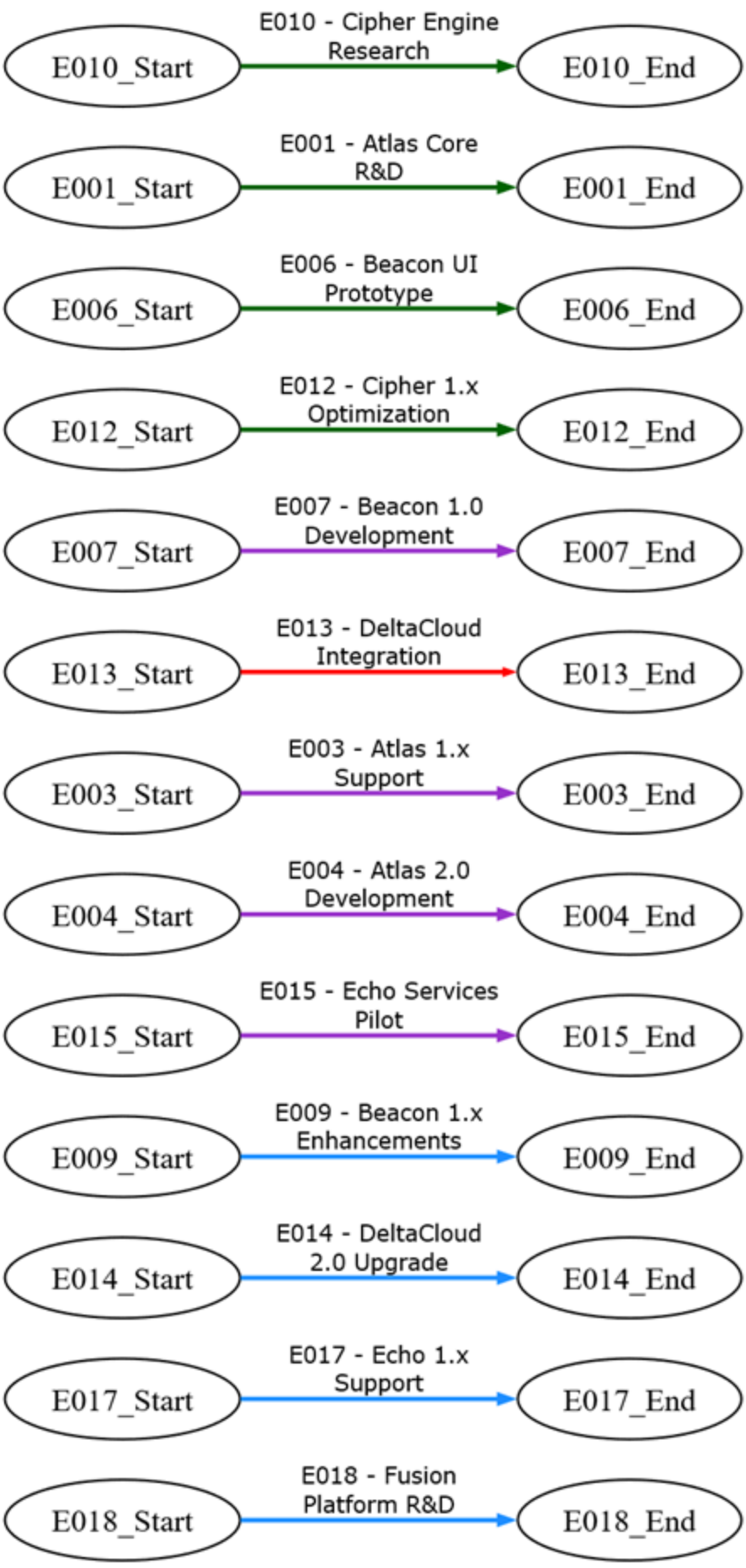
Step 4 — Connect Multi-Quarter Events

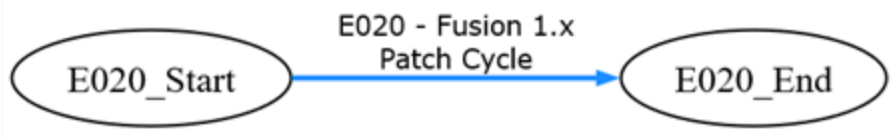
Many events span multiple quarters or even multiple years. To represent this, the roadmap creates edges from each event's start node to its end node.

- These edges carry the event's name as a label
- They use the event's status to determine styling
- A split length of 20 characters is applied to improve readability when labels are long.

```sql
SELECT [EventID] & '_Start'            AS [Item],
       [EventID] & '_End'              AS [Related Item],
       [EventID] & ' - ' & [EventName] AS [Label] ,
       [Status]                        AS [Style Name],
       20                              AS [SPLIT LENGTH]
FROM   [timeline$]
WHERE  [StartYear] IS NOT NULL
AND    [EndYear]   IS NOT NULL
AND    [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter]
ORDER BY [StartYear]    DESC,
         [StartQuarter] DESC
```

This produces a clear visual representation of duration: long events will stretch across the timeline, while shorter events will appear more compact.


E010 - Cipher Engine Research
E010_Start
E010_End
E001 - Atlas Core R&D
E001_Start
E001_End
E006 - Beacon UI Prototype
E006_Start
E006_End
E012 - Cipher 1.x Optimization
E012_Start
E012_End
E007 - Beacon 1.0 Development
E007_Start
E007_End
E013 - DeltaCloud Integration
E013_Start
E013_End
E003 - Atlas 1.x Support
E003_Start
E003_End
E004 - Atlas 2.0 Development
E004_Start
E004_End
E015 - Echo Services Pilot
E015_Start
E015_End
E009 - Beacon 1.x Enhancements
E009_Start
E009_End
E014 - DeltaCloud 2.0 Upgrade
E014_Start
E014_End
E017 - Echo 1.x Support
E017_Start
E017_End
E018 - Fusion Platform R&D
E018_Start
E018_End



Step 5 — Handle Single-Quarter Events

Events that start and finish in the same quarter require special handling. Instead of drawing a start-to-end edge, the roadmap connects the event's start node to itself.

```sql
SELECT [EventID] & '_Start'            AS [Item],
       [EventID] & '_Start'            AS [Related Item],
       [EventID] & ' - ' & [EventName] AS [Label] ,
       [Status]                        AS [Style Name],
       20                              AS [SPLIT LENGTH]
FROM   [timeline$]
WHERE  [StartYear] IS NOT NULL
AND    [EndYear]   IS NOT NULL
AND    [StartYear] & [StartQuarter] = [EndYear] & [EndQuarter]
ORDER BY [StartYear]    ASC,
         [StartQuarter] ASC
```

This preserves the visual semantics of "this event occurs entirely at this point in time" while still allowing the event to be styled and labeled consistently.

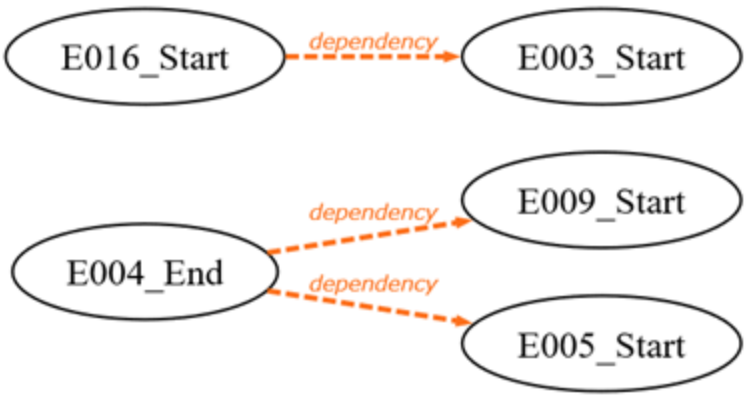
Step 6 — Add Dependency Edges

Roadmaps often include dependencies: one event must finish before another can begin. These relationships are drawn by joining the dependencies worksheet to the timeline data

and determining whether the dependency should originate from the event's start or end node.

```sql
SELECT
    'dependency' AS [Label],
    SWITCH(
        t.[StartYear] & t.[StartQuarter] = t.[EndYear] & t.[EndQuarter],
            d.[Must Complete] & '_Start',
        True,
            d.[Must Complete] & '_End'
    ) AS [Item],
    d.[Before the Start Of] & '_Start' AS [Related Item],
    'Dependency' AS [Style Name]
FROM
    [dependencies$]        AS d
    INNER JOIN [timeline$] AS t
        ON t.[EventID] = d.[Must Complete]
WHERE
    d.[Must Complete]       IS NOT NULL
AND d.[Before the Start Of] IS NOT NULL;
```

The resulting edges are styled distinctly so dependencies stand out from duration edges and timeline structure.



Step 7 — Group Events by Year Using Iteration

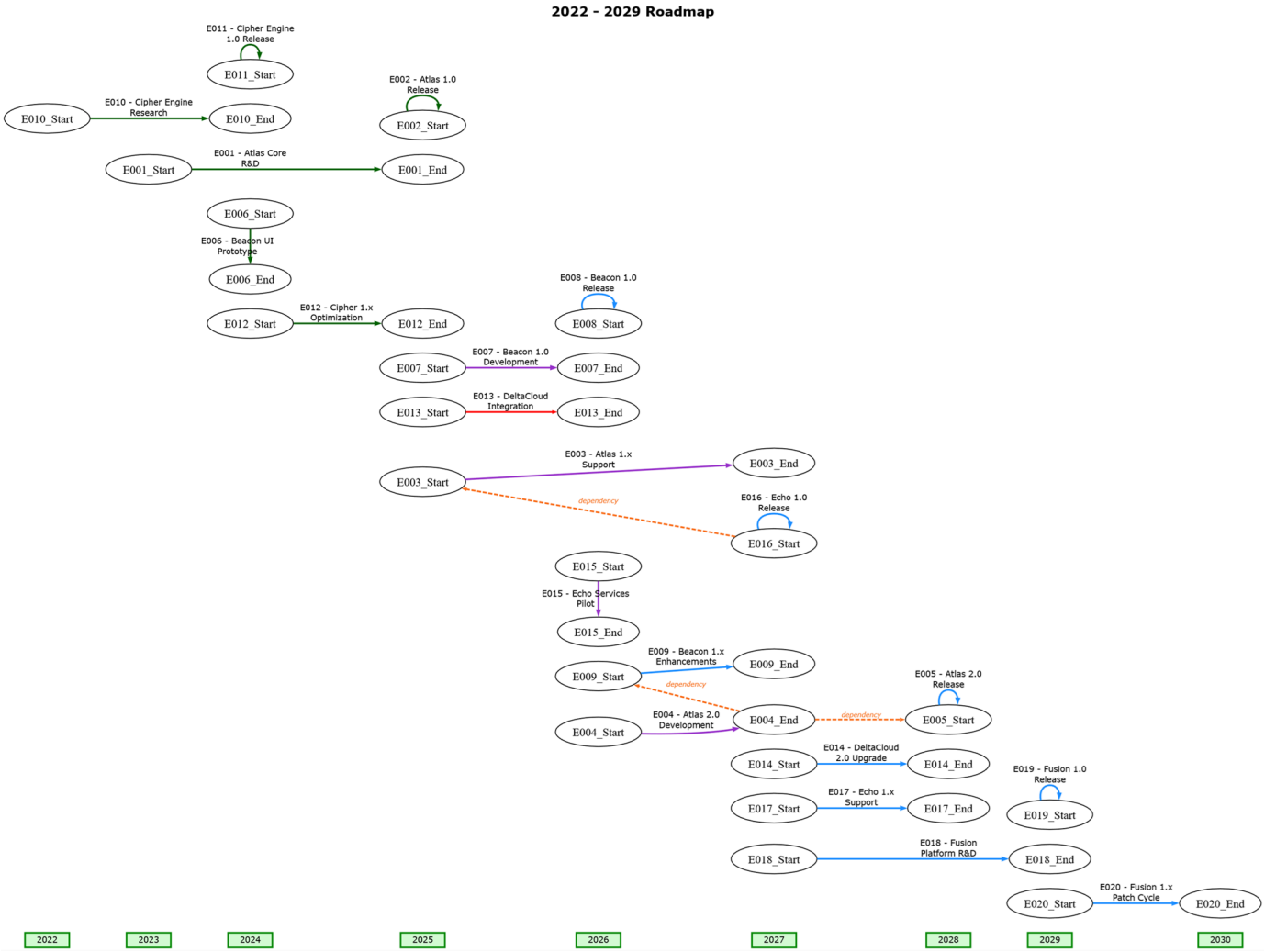
To keep the roadmap readable, all events belonging to the same year are placed on the same rank. This is accomplished using **iteration** and **UNION**. A normal query would place all items into a single subgroup, but iteration allows Relationship Visualizer to create one subgroup per year.

```sql
SELECT TRUE AS [ITERATE],
  'SELECT DISTINCT [StartYear] AS [ID] FROM [timeline$]'
  AS [SQL FOR ID],

  'SELECT TRUE AS [CREATE RANK], ''same'' AS [RANK], [StartYear]            AS [ITEM] F
   WHERE [StartYear] = {ID} AND [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter
      UNION
   SELECT TRUE AS [CREATE RANK], ''same'' AS [RANK], [EventID] & ''_Start'' AS [ITEM] F
   WHERE [StartYear] = {ID} AND [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter
      UNION
   SELECT TRUE AS [CREATE RANK], ''same'' AS [RANK], [EventID] & ''_Start''  AS [ITEM]
   WHERE [StartYear] = {ID} AND [StartYear] & [StartQuarter] = [EndYear] & [EndQuarter]
      UNION
   SELECT TRUE AS [CREATE RANK], ''same'' AS [RANK], [EndYear]              AS [ITEM] F
   WHERE [EndYear]   = {ID} AND [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter
      UNION
   SELECT TRUE AS [CREATE RANK], ''same'' AS [RANK], [EventID] & ''_End''   AS [ITEM] F
   WHERE [EndYear] = {ID} AND  [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter]
  '
  AS [SQL FOR DATA]
```

For each year:

- The year node is added to the subgroup
- All event start nodes for that year are added
- All event end nodes for that year are added (when appropriate)

This ensures that events align vertically with the year in which they occur, producing a clean, structured layout.



Step 8 — Style the Event Nodes

Finally, the roadmap applies styles to the event start and end nodes. Events that span multiple quarters receive distinct **“Start”** (round) and **“Finish”** (square) styles, while single-quarter events receive a unified **“Same”** (hexagon) style.

Start Quarter


```sql
SELECT [EventID] & '_Start'                 AS [Item],
       [StartYear] & '\n' & [StartQuarter] AS [Label],
       'Start'                              AS [Style Name]
```

```
FROM   [timeline$]
WHERE  [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter]
```

End Quarter

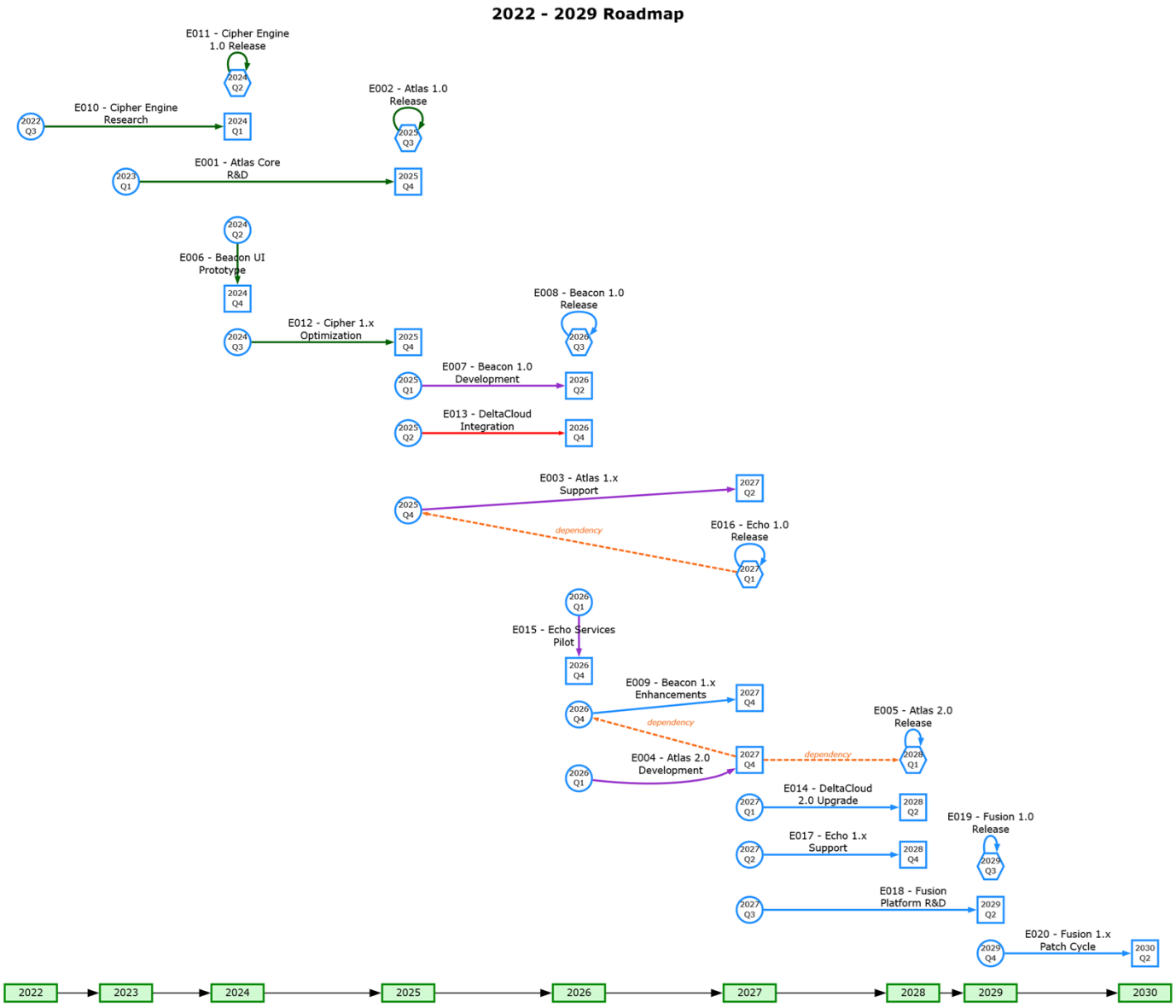

```sql
SELECT [EventID] & '_End'                AS [Item],
       [EndYear] & '\n' & [EndQuarter]   AS [Label],
       'Finish'                          AS [Style Name]
FROM   [timeline$]
WHERE  [StartYear] & [StartQuarter] <> [EndYear] & [EndQuarter]
```

Same Quarter


```sql
SELECT [EventID] & '_Start'                AS [Item],
       [StartYear] & '\n' & [StartQuarter] AS [Label],
       'Same'                              AS [Style Name]
FROM   [timeline$]
WHERE  [StartYear] & [StartQuarter] = [EndYear] & [EndQuarter]
```

Node labels include both the year and quarter, making it easy to see exactly when each event begins and ends.

The finished, complete roadmap appears as:



Try it Yourself

This example is included in the samples in the Relationship Visualizer zip file in the directory `18 - Using SQL - Timeline`.

Summary

This roadmap example demonstrates how iteration and enumeration work together to produce a rich, data-driven visualization:

- Enumeration creates the continuous timeline backbone.
- Iteration groups events by year for clean alignment.
- Event edges show duration.
- Dependency edges show sequencing.
- Styling layers bring clarity and structure.

By combining these features, Relationship Visualizer turns a basic event list into a fully navigable roadmap that reveals timing, duration, dependencies, and the overall flow of work. Healthy dependencies read naturally from left to right, while problematic or mis-aligned relationships surface instantly as right-to-left arrows, making potential issues easy to spot. Because all event dates live in a simple worksheet, the entire roadmap can be updated instantly—eliminating the tedious, error-prone task of redrawing timelines by hand.

SQL Syntax Reference

Structured Query Language (SQL) is one of the primary languages used to retrieve and manipulate data in relational systems. Microsoft Excel is often used in a similar fashion, storing information in worksheets that resemble database tables: columns with headers and many rows of data beneath them.

Excel supports SQL through its own dialect. Earlier versions relied on the Jet engine developed for Microsoft Access, while modern versions (beginning with Office 2007 and the `.xlsx` file format) use the ACE engine. Neither engine fully conforms to ANSI SQL standards, and both include unique behaviors and limitations that differ from traditional database systems.

This page consolidates information from many sources to provide a clear reference for the SQL syntax and functions supported by Microsoft Excel's SQL engine.

Excel SQL Statement Structure

Before diving into individual clauses and functions, it helps to understand how a SQL query is conceptually organized. Excel's ACE SQL engine follows the same logical evaluation order used by most relational database systems.

A SQL query can be viewed as a pipeline, where each stage refines or transforms the data.

- **SELECT — What columns to return**

  `SELECT` defines the output: which fields appear, in what order, and with what expressions, aliases, or calculations.

- **FROM — What table or range to read from**

  `FROM` identifies the worksheet or cell range that acts as the source dataset.

- **JOIN — How to combine tables.**

`JOIN` adds additional worksheets by matching rows across tables. JOINs expand the dataset horizontally (more columns).

- **WHERE — Which rows to keep**

  `WHERE` filters individual rows based on conditions. This is row-level filtering applied before grouping.

- **GROUP BY — How to group rows**

  `GROUP BY` aggregates rows into groups based on shared values.

- **HAVING — Which groups to keep**

  `HAVING` filters groups after aggregation, using conditions on aggregate values.

- **ORDER BY — How to sort the results**

  `ORDER BY` sorts the final output rows in ascending or descending order.

- **UNION / UNION ALL — How to combine result sets**

  `UNION` / `UNION ALL` stacks the results of multiple `SELECT` statements vertically.

  - `UNION` removes duplicates
  - `UNION ALL` preserves duplicates.

Understanding this flow helps you:

- Predict how Excel SQL will interpret your query
- Avoid errors caused by using clauses in the wrong context
- Write cleaner, more intentional SQL
- Recognize the difference between row-level, group-level, and query-level operations
- Understand why JOIN and UNION solve different problems

Referencing Data

The sections below describe how Excel SQL identifies and interprets the basic elements of a worksheet.

Before you can filter rows, join tables, or format results, SQL must understand where your data lives, how columns are named, how literal values are written, and how missing values behave. These conventions form the foundation for every query you write in Excel SQL.

The topics that follow explain:

- how to reference worksheets and cell ranges as tables
- how column names and data types are determined
- how to assign temporary names with aliases
- how to write and manipulate string values
- how Excel SQL represents and evaluates null values

Understanding these rules ensures that later clauses such as `WHERE`, `JOIN`, `GROUP BY`, and `ORDER BY` behave predictably and return the results you expect.

How to reference worksheets and cell ranges as tables

Tables are represented by Excel worksheets. A table is specified as the worksheet name followed by a `$` character. The square bracket characters `[` and `]` are used as delimiters, allowing worksheet names that contain spaces or special characters to be referenced safely.

In Excel SQL, table names follow this pattern:

Syntax:

```sql
[worksheetname$]
```

Example:

```sql
[Past Invoices$]
```

The example above refers to an open-ended Worksheet named `Past Invoices`.

It is also possible to use a portion of a worksheet, and not the whole worksheet. To use a partial worksheet, specify a cell range using standard Excel range notation. For example:

Example:

```sql
[Past Invoices$B10:G700]
```

Refers to to range of cells `B10:G700` within the *Past Invoices* Worksheet.

How column names and data types are determined

Column names can refer to the Excel heading for the column, such as A, B, C or the column heading in the first row of the table (or table range). Column names are limited to 64 characters. Column names containing blanks should also be enclosed in square brackets as in the following example. It is generally a good habit to always delimit column names with the square brackets.

Example:

```sql
[Customer Name]
```

The data type of a column is determined by the ACE engine by scanning the values in the first 8 rows in that column. If the scanned rows are blank then ACE assumes a String data type. If a column is determined to be numeric, then any rows with alpha characters in this column will be returned as Null.

How to assign temporary names with aliases

Aliases are used to temporarily assign a different name to a table or column heading using an `AS` clause. Basically aliases are created to make column names more readable. The `WHERE`, `ORDER BY`, `GROUP BY`, `HAVING` and `JOIN` clauses support aliases.

SQL alias syntax for columns follows the syntax format:

Syntax:

```sql
SELECT [column_name] AS column_alias_name FROM [worksheet$]
```

SQL alias syntax for tables follows the syntax format:

Syntax:

```sql
SELECT [column_name] FROM [worksheet$] AS table_alias_name
```

The following example shows multiple ways in which an alias can be expressed within an Excel SQL statement.

Example:

```sql
SELECT [country] AS CC, [Units Sold]
FROM [Sales$] AS sales
WHERE [sales.segment] = 'Government' AND [sales].[CC] = 'US'
```

How to write and manipulate string values

Most SQL statements involve the use of **string values** as criteria in a WHERE clause, or will return string values in the column results of a SELECT clause.

Strings are denoted in SQL using the single quotation mark `'` character, as opposed to the double quote `"` character used in VBA and other programming languages. A String value such as `'Central Region'` is properly formatted for use as selection criteria, or as a value to be inserted.

Strings can be concatenated using the ampersand `&` character. For example, the statement `'Central' & ' ' & 'Region'` equates to the String value `'Central Region'`.

Microsoft Excel SQL provides numerous String functions that perform operations on an input String and return a String or numeric value result. These functions can be combined to create unique input or output values depending upon the SQL statement. A table of commonly used String Functions is provided at the end of this document as reference.

How Excel SQL represents and evaluates null values

A **Null Value** is a value that is unavailable, unassigned, unknown or inapplicable. If a row lacks the data value for a particular column, that value is said to be null or to contain a null. A Null Value is like a character string of length zero, however you should never use a Null Value to represent a value of zero.

Since Microsoft Excel SQL allows null values, it is your responsibility to take them into consideration in your SQL statements. Null Handling is described in greater detail later in this document.

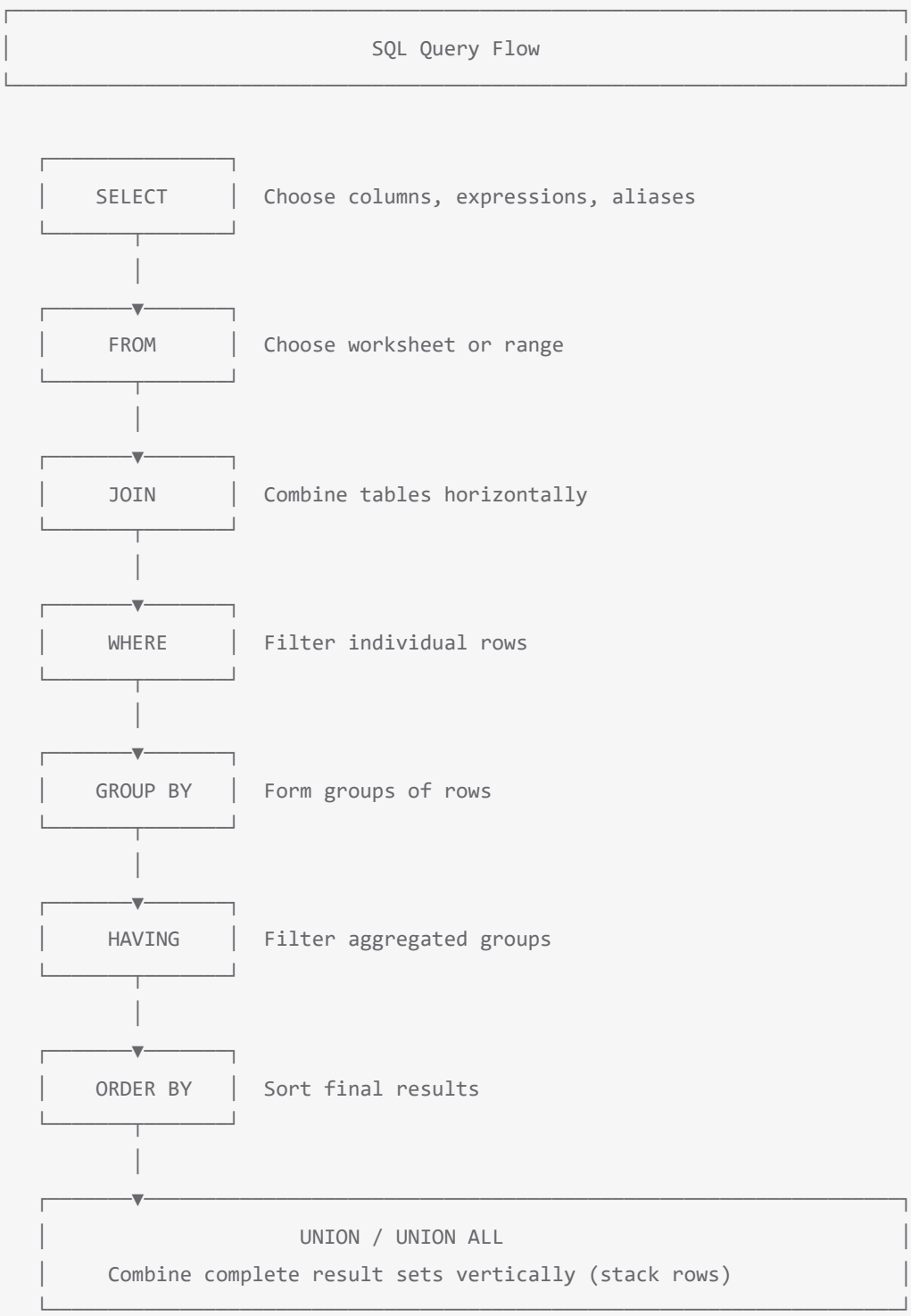
Querying Data

Microsoft Excel SQL queries support `FROM`, `JOIN`, `WHERE`, `GROUP BY`, `HAVING`, and `ORDER BY` clauses.

Excel SQL `SELECT` statements conform to the following pattern:

```sql
SELECT [DISTINCT] [TOP [PERCENT] n] * | column1, column2 [AS alias2], ...
[FROM worksheet]
[JOIN worksheet ON condition]
[WHERE condition]
[GROUP BY column1, column2, ...]
[HAVING condition]
[ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...]
```

SQL Query Flow



SELECT clause

The `SELECT` clause is used to execute a query to select data from a Microsoft Excel worksheet.

`SELECT *`

A query that selects all rows and columns from the Excel file.

Example:

```sql
SELECT * FROM [Sales$];
```

In this example, the query fetches all rows and columns in the Sales worksheet. You can query against different sheets in a common Excel file using this syntax.

`SELECT TOP n`

The `TOP` option specifies how many rows to return.

Example:

```sql
SELECT TOP 5 [Quantity], [Name], [Price] FROM [Sales$];
```

Here a maximum of 5 rows will be returned.

`SELECT TOP n PERCENT`

The `TOP PERCENT` option specifies the percentage of the complete result set to return.

Example:

```sql
SELECT TOP 25 PERCENT [Quantity], [Name], [Price] FROM [Sales$];
```

Here the first 25% of the result set rows will be returned. If the query result set contains 400 rows, then the top 100 rows will be returned.

`SELECT` column list

Create a query that selects specific columns from the Excel file.

Example:

```sql
SELECT [Quantity], [Name], [Price] FROM [Sales$];
```

`SELECT` from a range of cells

Limit your query to a specific cell range.

Example:

```sql
SELECT [Quantity], [Name], [Price] FROM [Sales$A1:E101];
```

In this example, we do not impose any limitations on the values themselves. However, we direct the query to look only at a Range of cells (A1 through E101). Note that the cell range is specified after the dollar sign in the table name, using the colon between the first cell and the final cell in the range.

`SELECT DISTINCT`

A column can contain duplicate values, and to list the distinct values, use the `SELECT DISTINCT` clause. The `DISTINCT` clause can be used to return only distinct values from a set of records.

Example:

```sql
SELECT DISTINCT [Name], [Price] FROM [Sales$A1:E101];
```

Aggregate functions

An aggregate function performs a calculation on a set of values, and returns a single value. Except for `COUNT(*)`, aggregate functions ignore null values. Aggregate functions are often used with the `GROUP BY` clause of the `SELECT` statement.

Function	Description
`AVG`	Returns the average (mean) value. It ignores null values. If no rows are selected, the result is NULL.
`COUNT(*)`	Returns the number of rows in a specified table, and it preserves duplicate rows. It counts each row separately. This includes rows that contain null values.
`MAX`	Returns the highest value.
`MIN`	Returns the lowest value.
`STDEV`	Returns the statistical standard deviation of all values in the specified expression.
`STDEVP`	Returns the statistical standard deviation for the population for all values in the specified expression.
`SUM`	Returns the sum of all values.
`VAR`	Returns the statistical variance of all values in the specified expression.
`VARP`	Returns the statistical variance for the population for all values in the specified expression.

Example:

```sql
SELECT COUNT(*)
FROM [Sales$];
```

In this example the number of rows in the Sales worksheet is returned.

Example:

```sql
SELECT SUM([Units Sold]), AVG([Units Sold])
FROM [Sales$]
```

In this example the total number of units sold, and the average quantity of units sold is returned.

Arithmetic

It is possible within a `SELECT` statement to run mathematical operations on two expressions of one or more data types.

Arithmetic Operators

Arithmetic operators can be used to perform mathematical operations against query results. They are:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
MOD	Modulo, returns the integer remainder of a division

Example:

```sql
SELECT [units sold] MOD 10 AS UNITS_SOLD_MOD_10
```

```
    FROM [Sales$]
```

Here the number of units sold is calculated as modulo 10 and the result is expressed using the alias `UNITS_SOLD_MOD_10`.

Example:

```sql
SELECT [sale price], ([sale price] * 1.06) AS [sale price plus tax]
FROM [Sales$]
```

In this example the sale price is multiplied by 1.06 to reflect a 6% sales tax, and is expressed using the alias sale price plus tax.

Mathematical Functions

The following scalar functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value:

Function	Description
`ABS`	A mathematical function that returns the absolute (positive) value of the specified numeric expression. (ABS changes negative values to positive values. ABS has no effect on zero or positive values.)
`COS`	A mathematical function that returns the trigonometric cosine of the specified angle - measured in radians - in the specified expression.
`EXP`	Returns the exponential value of the specified expression.
`LOG`	Returns the natural logarithm of the specified expression.
`ROUND`	Returns a numeric value, rounded to the specified length or precision.
`SIN`	Returns the trigonometric sine of the specified angle, in radians, and in an approximate numeric, float, expression.
`TAN`	Returns the tangent of the input expression.

Example:

```sql
SELECT EXP(LOG(20)), LOG(EXP(20))
```

Here the SQL statement returns the exponential value of the natural logarithm of 20 and the natural logarithm of the exponential of 20. Because these functions are inverse functions of one another, the return value in both cases is 20.

Conditionals

Conditionals are used to conditionally modify results.

`IIF()` function

The `IIF()` function provides simple conditional logic inside a `SELECT` clause. It evaluates an expression and returns one of two values depending on whether the expression is true or false.

Syntax:

```sql
IIF(expression, truepart, falsepart)
```

- **expression:** must evaluate to TRUE or FALSE
- **truepart:** returned when the expression is TRUE
- **falsepart:** returned when the expression is FALSE

Example:

```sql
SELECT
    [Product],
    [Units Sold],
    [Sale Price],
```

```
    IIF([Sale Price] < 10.00, 'SPECIAL!', '') AS [Flag]
FROM [Sales$]
```

This returns `'SPECIAL!'` when the sale price is below 10.00; otherwise it returns an empty string.

Example (handling nulls):

```sql
SELECT
  IIF(IsNull([Units Sold]), 0, [Units Sold]) AS [Units Sold Clean]
FROM [Sales$]
```

Null Handling

Under some conditions, such as when a cell is has no data, Excel returns a Null value. You can test whether a cell is null using the `IsNull(expression)` function. `IsNull()` returns a Boolean true value (=-1) if the argument is null, and a Boolean false (=0) if the argument is not null.

A simple example is:

Example:

```sql
SELECT IsNull([Units Sold])
FROM [Sales$]
```

`IIF()` is often combined with `IsNull()` to enforce predictable behavior when aggregating or formatting values. The `IsNull()` function can be combined with the `IIF()` syntax to return a specific value in cases where a null is found in a column, and the actual column value where the value is not null.

`SWITCH()` function

The `SWITCH()` function evaluates multiple conditions in order and returns the value associated with the first condition that is TRUE. It is the ACE SQL equivalent of a multi-branch CASE expression.

Syntax:

```sql
SWITCH(
    condition1, value_if_true1,
    condition2, value_if_true2,
    ...,
    True, default_value
)
```

Key behaviors:

- Conditions are evaluated **left to right**
- The first TRUE condition determines the return value
- If no condition is TRUE and no default is provided, the result is `NULL`
- All expressions are evaluated, so avoid expressions that may error (e.g., division by zero)

Example:

```sql
SELECT
  [Score],
  SWITCH(
      [Score] >= 90, 'A',
      [Score] >= 80, 'B',
      [Score] >= 70, 'C',
      True,          'D'
  ) AS [Grade]
FROM [Sales$]
```

This assigns a letter grade based on score ranges.

Example (categorizing values):

```sql
SELECT
  [Region],
  SWITCH(
      [Region] = 'North', 'Domestic',
      [Region] = 'South', 'Domestic',
      [Region] = 'Canada', 'International',
      True,               'Other'
  ) AS [Region Class]
FROM [Sales$]
```

Aggregate Functions + Conditionals + Null Handling

Conditional tests are very useful in situations where you are aggregating data which may have `NULL` values. You can ensure predictable behavior by using `IFF` to return a consistent value within an aggregate function when `NULL` is encountered. For example, if you want `NULL` to be treated a zero, you write the SQL as:

Example:

```sql
SELECT MIN(IIf(IsNull([Units Sold]), 0, [Units Sold]))
FROM [Sales$]
WHERE [country] = 'Canada'
```

Formatting date values

`FORMATDATETIME` function

The `FORMATDATETIME` function has a set of fixed options for formatting, shown below.

Format Code	Description
0	Display a date and/or time. Date parts are displayed in short date format. Time parts are displayed in long time format. For example, "1/1/2014"

Format Code	Description
1	Display a date using the long date format specified in your computer's regional settings. For example, "Wednesday, January 1, 2014"
2	Display a date using the short date format specified in your computer's regional settings. For example, "1/1/2014".
3	Display a time using the time format specified in your computer's regional settings. For example "12:00:00 AM".
4	Display a time using the 24-hour format (hh:mm). For example, "00:00".

Example:

```sql
SELECT
[Date],
FORMATDATETIME([Date],0) AS [FormatDateTime0],
FORMATDATETIME([Date],1) AS [FormatDateTime1],
FORMATDATETIME([Date],2) AS [FormatDateTime2],
FORMATDATETIME([Date],3) AS [FormatDateTime3],
FORMATDATETIME([Date],4) AS [FormatDateTime4]
FROM [Sales$]
```

`FORMAT` function

If you need a more flexible set of formatting options, the `FORMAT` function takes a format template string:

String	Description
d	display the day as a number without a leading zero (1 - 31).
dd	Display the day as a number with a leading zero (01 - 31).
ddd	Display the day as an abbreviation (Sun - Sat).

String	Description
m	Display the month as a number without a leading zero (1 - 12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01 - 12). If m immediately follows h or
mmm	Display the month as an abbreviation (Jan - Dec).
mmmm	Display the month as a full month name (January - December).
oooo	The same as mmmm, only it's the localized version of the string.
y	Display the day of the year as a number (1 - 366).
yy	Display the year as a 2-digit number (00 - 99).
yyyy	Display the year as a 4-digit number (100 - 9999).

Example:

```sql
SELECT [Date], FORMAT([Date],"yyyy-mmm-dd") AS [formatted_date],
FROM [Sales$]
```

In the example above, `[Date]` will be returned as an absolute number such as 41640, and `[formatted_date]` will be returned as a string such as 2014-Jan-01 as specified by the format pattern.

Formatting numbers

The `FORMAT` function can also be used to format date formats other than dates. Assume we have a query which calculates sales tax as 6% of the sale price. The query would look as follows:

Example:

```sql
SELECT
  [Gross Sales],
  [Gross Sales]*0.06 as [Sales Tax]
FROM [Sales$]
```

The results of this query would have varying decimal places of 0 or more. To always return 2 decimal places we can modify the query as follows:

Example:

```sql
SELECT
  [Gross Sales],
  FORMAT([Gross Sales]*0.06,'0.00') as [Sales Tax]
FROM [Sales$]
```

The `[Sales Tax]` column is formatted to two decimal places. Note that the formatting code must be enclosed in single quotes, not double quotes. Note also that the format function returns a string type, so if you want to do math with a formatted value, you'll have to later convert it to a numerical type with a type conversion function.

An alternate way to format numbers is to use the named numeric formats. Since we were formatting the sales tax with two decimal places to represent currency, we could also use the 'Currency' named format. This format will display the number with the currency symbol, the thousand separator, and if appropriate; display two digits to the right of the decimal separator. Output is based on system locale settings.

Example:

```sql
SELECT
  [Gross Sales],
  FORMAT([Gross Sales]*0.06,'Currency') as [Sales Tax]
FROM [Sales$]
```

Here the `[Sales Tax]` will be returned with an appearance such as $2,224.80.

For more information on the `FORMAT` function and a listing of format options for Numbers, Dates and times, Date and time serial numbers, and Strings, please refer to Microsoft's documentation of the FORMAT function ↗.

FROM clause

The `FROM` clause determines the specific dataset to examine to retrieve data. For Excel, this dataset would be indicated as a specific worksheet, with a `$` appended to the worksheet name.

Example:

```sql
SELECT * FROM [Sales$];
```

In this example the Sales worksheet is specified as `[Sales$]`

JOIN clause

There are different types of joins available in Excel SQL: `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `CROSS JOIN`, and `self joins`.

- `INNER JOIN` returns rows when there is a match on both tables.
- `LEFT JOIN` returns all rows from the left table even if there are no matches in the right table.
- `RIGHT JOIN` returns all rows from the right table even if there are no matches in the left table.
- `CROSS JOIN` (or `CARTESIAN JOIN`) returns the Cartesian product of the sets of records from two or more joined tables.
- `Self join` is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement. You can find the syntax for the

different joins below.

INNER JOIN clause

The `INNER JOIN` keyword selects all rows from both tables as long as there is a match between the columns in both tables.

The `INNER JOIN` syntax follows this syntx pattern:

Syntax:

```sql
SELECT columns
FROM worksheet1 AS worksheet1_alias
INNER JOIN worksheet2 AS worsksheet2_alias
ON worksheet_alias1.column1 = worksheet2_alias.column2
```

The valid operator for the `ON` clause are `AND`, `OR`, `=`, `>`, `<`, `<>`, `>=`, `<=`, `!=`. In addition, nested joins are supported. Nested joins of more than two tables must be enclosed in parenthesis.

Example:

```sql
SELECT
    A.[Segment], A.[Country], A.[Product],
    B.[Units Sold], B.[SKU]
FROM [Products$] AS A
INNER JOIN [Sales$] AS B
ON A.[SKU] = B.[SKU]
```

LEFT JOIN clause

The `LEFT JOIN` clause returns all rows from the left table (worksheet1) with the matching rows in the right table (worksheet2). The result set returns no values in the right table when there is no match.

The `LEFT JOIN` syntax follows this pattern:

Syntax:

```sql
SELECT columns
FROM worksheet1 AS worksheet1_alias
LEFT JOIN worksheet2 AS worsksheet2_alias
ON worksheet_alias1.column1 = worksheet2_alias.column2
```

The valid operator for the `ON` clause are `AND`, `OR`, `=`, `>`, `<`, `<>`, `>=`, `<=`, `!=`. In addition, nested joins are supported. Nested joins of more than two tables must be enclosed in parenthesis.

Example:

```sql
SELECT
    A.[Segment], A.[Country], A.[Product],
    B.[Units Sold], B.[SKU]
FROM [Products$] AS A
LEFT JOIN [Sales$] AS B
ON A.[SKU] = B.[SKU]
```

RIGHT JOIN clause

The `RIGHT JOIN` clause returns all rows from the right table (worksheet2) with the matching rows in the left table (worksheet1). The result set returns no values in the left table when there is no match.

The `RIGHT JOIN` syntax follows this syntax pattern:

Syntax:

```sql
SELECT columns
FROM worksheet1 AS worksheet1_alias
```

```
RIGHT JOIN worksheet2 AS worksksheet2_alias
ON worksheet_alias1.column1 = worksheet2_alias.column2
```

The valid operator for the `ON` clause are `AND`, `OR`, `=`, `>`, `<`, `<>`, `>=`, `<=`, `!=`. In addition, nested joins are supported. Nested joins of more than two tables must be enclosed in parenthesis.

Example:

```sql
SELECT
    A.[Segment], A.[Country], A.[Product],
    B.[Units Sold], B.[SKU]
FROM [Products$] AS A
RIGHT JOIN [Sales$] AS B
ON A.[SKU] = B.[SKU]
```

CROSS JOIN clause

The `CROSS JOIN` (or `CARTESIAN JOIN`) returns the Cartesian product of the sets of records from two or more joined tables. It equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement. Each row in the first table is paired with all the rows in the second table. This happens when there is no relationship defined between the two tables.

The `CROSS JOIN` syntax follows this pattern:

Syntax:

```sql
SELECT column1, column2 FROM [worksheet1$] CROSS JOIN [worksheet2$]
```

Example:

```sql
SELECT A.[E_id], B.[P_id], A.[fname]
FROM [employee$] AS A
```

```
CROSS JOIN [project$] AS B
```

Self Join

`Self join` is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The Self Join syntax follows this pattern:

Syntax:

```sql
SELECT column1, column2
FROM [worksheet1$] AS alias1, [worksheet1$] AS alias2
WHERE alias1.column1 = alias2.column2
```

Example:

```sql
SELECT A.[E_id], B.[E_id]
FROM [employee$] AS A, [employee$] AS B
WHERE A.[E_id] = B.[Mgr_id]
```

WHERE clause

The `WHERE` clause, sometimes called the predicate, states the qualifying conditions for a query. You can combine arithmetic operators and the comparison operators in the `WHERE` clause. Multiple conditions can be joined by the `AND` and `OR` clauses, optionally surrounded by (parentheses) to group them. Only the records that satisfy the specified criteria are returned by the query.

`WHERE` comparison operators

Comparison operators test whether two expressions are the same. The following list contains the comparison operators supported in a `WHERE` clause. `WHERE` conditions follow the syntax:

`WHERE` column `operator` predicate

Operator	Description
`=`	Equals operator. Compares the equality of two expressions.
`!=`	Not Equal To operator. Compares two expressions and returns TRUE if the left operand is not equal to the right operand; otherwise, the result is FALSE.
`<>`	Not Equal To operator. Compares two expressions and returns TRUE if the left operand is not equal to the right operand; otherwise, the result is FALSE.
`>`	Greater Than operator. Compares two expressions and returns TRUE if the left operand has a value higher than the right operand; otherwise, the result is FALSE.
`>=`	Greater Than or Equal To operator. Compares two expressions and returns TRUE if the left operand has a value greater than or equal to the right operand; otherwise, it returns FALSE.
`<=`	Less Than or Equal To operator. Compares two expressions and returns TRUE if the left operand has a value lower than or equal to the right operand; otherwise, it returns FALSE.
`<`	Less Than operator. Compares two expressions and returns TRUE if the left operand has a value lower than the right operand; otherwise, the result is FALSE.
`IS NULL`	A Null value is a value that is unavailable, unassigned, unknown or inapplicable. This operator is used to determine if a field doesn't contain data.
`IS NOT NULL`	This operator is used to determine of a field does contain data.

When specifying the condition, value must be an exact match of the column value in the worksheet.

String value comparisons are case sensitive. You can use the `UCASE()` (i.e. upper case) or `LCASE()` (i.e. lower case) function to perform case insensitive comparisons.

`WHERE` logical operators

Comparison operators test whether two expressions are the same. The following list contains the logical operators supported in a `WHERE` clause.

Operator	Description
`ALL`	Returns TRUE when all of the subquery values meet the condition.
`AND`	Combines two Boolean expressions and returns TRUE when both expressions are TRUE. *Syntax:* condition `AND` condition
`ANY`	Returns TRUE when any of the subquery values meet the condition. *Syntax:* condition `ANY` condition
`BETWEEN`	Specifies a range to test. Returns TRUE when the operand is within the range of comparisons. *Syntax:* column `BETWEEN` value1 `AND` value2
`EXISTS`	Specifies a subquery to test for the existence of rows. Returns TRUE when the subquery returns one or more records. *Syntax:* column `EXISTS`
`IN`	Determines whether a specified value matches any value in a subquery or a list. Returns TRUE when the operand is equal to one of a list of expressions.

Operator	Description
	Syntax: column `IN` (value [, value ...])
`LIKE`	Determines whether a specific character string matches a specified pattern. Returns TRUE when the operand matches a pattern. *Syntax:* column `LIKE` like_expression
`NOT`	Negates a Boolean input (it reverses the value of any Boolean expression). It therefore returns TRUE when the expression is FALSE. *Syntax:* column `NOT EXISTS` column `NOT BETWEEN` value1 `AND` value2
`OR`	Combines two conditions. Returns TRUE when either of the conditions is TRUE. *Syntax:* condition `OR` condition
`SOME`	Same as ANY. Returns TRUE when any of the subquery values meet the condition.

Boolean Precedence

When using `AND` and `OR` to specify multiple conditions, use (parentheses) to group the conditions. If no parentheses are specified, the conditions specified with `AND` are evaluated together.

Syntax:

The expression

```sql
condition1 AND condition2 AND condition3 OR condition4
```

is equivalent to

```
(condition1) AND (condition2) AND (condition3 OR condition4)
```

Syntax:

The expression

```
condition1 AND condition2 OR condition3 AND condition4
```

is equivalent to

Syntax:

```
(condition1) AND (condition2 OR condition3) AND (condition4)
```

`WHERE` with multiple criteria

Use a `WHERE` clause in your query to filter your Excel data.

Example:

```sql
SELECT [Quantity], [Name], [Price] FROM [Sales$]
WHERE [Sale ID] >= 23 AND [Sale ID] <= 28;
```

In this example, we limit our result set to records whose `[Sale ID]` is >= 23 and < 28. The syntax for column names in the `WHERE` clause uses square brackets, as we saw previously

`WHERE` with string values

According to the SQL standard, the text delimiter in SQL is the single quotation mark (`'`). Use the single quote character (`'`) in your query to denote strings to match against your Excel data.

Example:

```sql
SELECT * FROM [Sales$]
WHERE [Country] = 'Canada'
```

In this example, we limit our result set to records where the `[Country]` value matchs the string Canada.

String Functions allow you to create a query using a string function in the `WHERE` clause, or to modify the column results of the query.

Example:

```sql
SELECT UCASE([Name]),[Price]
FROM [Sales$]
```

In the example above, the VBA `UCASE()` function is being used to convert the `NAME` values to upper case.

Example:

```
SELECT [Sale ID], [Sale Date], [Quantity], [Name], [Price]
FROM [Sales$]
WHERE MID([Name],1,4) = 'NYNY'
```

Here the VBA `MID()` function is being used to limit our results to the records which contain the string 'NYNY' in the first four characters of the `NAME` column.

Example:

```sql
SELECT [Name], [Price], 'Central' AS [REGION]
FROM [Sales$]
```

Here a constant value of 'Central' will be included with every row returned under the column name [REGION].

In Excel SQL the ampersand `&` symbol is used to peform string concatenation. In the example below the table values of `[Street]`, `[City]`, `[State]`, and `[Zip Code]` will be concatentated along with formatting strings to build a single address string returned as the column name `[Address]`.

Example:

```sql
SELECT
[Street] & ', ' & [City], & ' ' & [State] & ' ' & [Zip Code] & ' USA' AS [Address]
FROM [Sales$]
```

`WHERE` with date values

Date literals need to be enclosed in delimiters in SQL strings. The hash sign `#` acts as the delimiter for date values. Use the `#` character in your query to filter your Excel data by date.

Example:

```sql
SELECT * FROM [Sales$]
WHERE [Date] BETWEEN #9/1/2013# AND #12/31/2013#
```

In this example, we limit our result set to records between the dates of September 1, 2013 and December 31, 2013 by sepcifying the date in the local format and wrapping the value with `#` characters so a date comparison is performed against the date serial number instead of treating the date as a string value.

`WHERE` with delimiters in text literals

Consider the case where you have a text field in your table and you want to query to query it within the WHERE condition of a query. In the query the criteria values themselves, however, contain the text delimiter, i.e. the single quote. This would be the case when searching for the name O'Brien. If you a text literal of 'O'Brien' inside your SQL statement it will cause a syntax error. The solution is to double the single quote inside the text literal and it will be treated as just one single quote inside the literal.

Example:

```sql
SELECT * FROM [Sales$] WHERE [CustomerName] = 'Martha O''Brian'
```

GROUP BY clause

The `GROUP BY` clause is used in combination with aggregate functions (e.g. SUM) to group the results by one or more columns.

Example:

```sql
SELECT DISTINCT [Name], SUM([Quantity])
FROM [Sales$]
GROUP BY [Name]
```

`GROUP BY` must include ALL the non-aggregated `SELECT` expressions. The first column is not enough.

HAVING clause

The `HAVING` clause states the qualifying conditions for aggregated values. It is used in conjunction with aggregate functions to filter aggregated values.

A typical select statement using the `HAVING` clause will follow the syntax pattern:

Syntax:

```sql
SELECT [column1], aggregate_function([column2]) FROM [worksheet$]
GROUP BY [column1]
HAVING aggregate_function([column2]) comparison_operator value
```

as in the following example which returns the count of the number of orders associated with a name if the count is greater than 25.

Example:

```sql
SELECT [name], COUNT([orders])
FROM [Sales$]
GROUP BY [name]
HAVING COUNT([orders]) > 25
```

ORDER BY clause

`ORDER BY` is used to sort the results by one or more columns and sorts in ascending order by default. To sort in descending order, use the `DESC` keyword.

A typical SELECT statement using the ORDER BY clause will follow the pattern:

Syntax:

```sql
SELECT [column1], [column2]
FROM [worksheet$]
ORDER BY [column1] ASC | DESC, [column2] ASC | DESC
```

where `ASC` refers to Ascending Order (A-Z, 0-9), and `DESC` refers to Descending Order (Z-A, 9-0). For example:

Example:

```sql
SELECT *
FROM [Sales$]
ORDER BY [units sold] DESC
```

In this example all the rows and columns from the `Sales` worksheet are selected and returned in descending order as determined through the number of units sold.

The `ORDER BY` clause cannot be used on a column with mixed data types. If a column contains data with mixed data types, the data needs to be converted to one data type.

Note that the `ORDER BY` clause also cannot reference aliases, as you will receive the error `"No value given for one or more required parameters"`. Instead, use the ordinal index in the `SELECT` clause as follows to order the results:

Example:

```sql
SELECT [country] AS CC, [Units Sold]
FROM [Sales$] AS sales
WHERE [sales.segment] = 'Government'
ORDER BY 1
```

Combining Result Sets

The `UNION` and `UNION ALL` operators combine the results of two or more `SELECT` statements. Each `SELECT` must return the same number of columns, in the same order, with compatible data types.

UNION

`UNION` combines result sets and removes duplicate rows. Because duplicates must be compared and eliminated, `UNION` is slower than `UNION ALL`.

Example:

```sql
SELECT [Name], [Region]
FROM [Sales$]
UNION
SELECT [Name], [Region]
FROM [Archive$]
```

This returns distinct rows across both worksheets.

UNION ALL

`UNION ALL` combines result sets without removing duplicates.
It is faster and should be used when you want all rows preserved.

Example:

```sql
SELECT [Name], [Region]
FROM [Sales$]
UNION ALL
SELECT [Name], [Region]
FROM [Archive$]
```

This returns all rows, including duplicates.

`UNION ALL` - Generate values

Because ACE SQL does not support `VALUES()` or table literals, `UNION ALL` is often used to create small inline datasets.

Example:

```sql
SELECT 1 AS [Month]
UNION ALL SELECT 2
UNION ALL SELECT 3
UNION ALL SELECT 4
```

This produces a four-row result set containing the numbers 1–4.

`UNION ALL` - Expand rows

`UNION ALL` can also be used to create repeated or expanded rows when no numbers table exists.

Example:

```sql
SELECT [Product], [Quantity], 'Q1' AS [Quarter] FROM [Sales$]
UNION ALL
SELECT [Product], [Quantity], 'Q2' FROM [Sales$]
UNION ALL
SELECT [Product], [Quantity], 'Q3' FROM [Sales$]
UNION ALL
SELECT [Product], [Quantity], 'Q4' FROM [Sales$]
```

This produces four rows per product — one for each quarter.

String Functions

The following table summarizes the most commonly used String functions used in Microsoft Excel SQL statements.

Function	Description
`CHR`	Returns the character based on the ASCII value. The syntax is `CHR( ascii_value )` where `ascii_value` is the decimal value used to retrieve the character from the ASCII Table ↗. For example, `CHR(37)` would return `%` (i.e. the Percent Sign character).

`INSTR`	Returns the position of the first occurrence of a substring in a string. The syntax for the `INSTR` function in Microsoft Excel SQL is:
	
	`INSTR( string, substring )`
	
	where `string` is the string to search within, and `substring` is the substring which you want to find.
	
	For example, the SQL uery
	
	`SELECT [Segment], INSTR([Segment],'er') AS [erPosition] FROM [sheet1$]`
	
	returns the position of the letters 'er' in the value of the `Segment` column. If `Segment` were to contain the value 'Government', the function will returns a value of 4.

`LCASE`	Converts a string to all lowercase. The syntax for the `LCASE` function in Microsoft Excel SQL is: `LCASE( text )` where `text` is the string which you wish to convert to lower-case.

Function	Description
`LEFT`	Extract a substring from a string, starting from the left-most character. The syntax for the `LEFT` function in Microsoft Excel SQL is: `LEFT( text, number_of_characters )` where `text` is the string which you wish to extract from, and `number_of_characters` indicates the number of characters that you wish to extract starting from the left-most character.
`LEN`	Returns the length of the specified string. The syntax for the `LEN` function in Microsoft Excel SQL is: `LEN( text )` where `text` is the string that you wish to determine the length of.
`LTRIM`	Removes leading spaces from a string. The syntax for the `LTRIM` function in Microsoft Excel SQL is: `LTRIM( text )` where `text` is the string that you wish to remove leading spaces from.
`MID`	Extracts a substring from a string (starting at any position). The syntax for the `MID` function in Microsoft Excel SQL is: `MID( text, start_position, number_of_characters )` where `text` is the string which you wish to extract from, `start_position` indicates the position in the string that you will begin extracting from (the first position in the string is 1), and `number_of_characters` is the number of characters that you wish to extract.
`REPLACE`	Replaces a sequence of characters in a string with another set of characters. The syntax for the REPLACE function in Microsoft Excel SQL is: `REPLACE ( string1, find, replacement, [start, [count]] )`

Function	Description
	Where `string1` is the string to replace a sequence of characters with another set of characters, `find` is the string that will be searched for in `string1`, `replacement` is the string which will replace `find` in `string1`. For example, the SQL statement `SELECT REPLACE([Segment],' ','_') FROM [sheet1$]` would change all space characters in the `Segment` column values to underscores (a value such as 'Channel Partners' would be returned as 'Channel_Partners'). Parameter `start` is optional, and specifies the position in `string1` to begin the search. If the `start` parameter is omitted, the `REPLACE` function will begin the search at position 1. Parameter `count` is also optional, and specifies the the number of occurrences to replace. If the `count` parameter is omitted, the `REPLACE` function will replace all occurrences of `find` with `replacement`. If you wish to specify the `count` parameter, you must also specify the `start` parameter. For example, the SQL statement `SELECT REPLACE([Segment],'e','X',1,2)FROM [sheet1$]` would change the first 2 occurences of the letter 'e' in the `Segment` values to the letter 'X' (A value such as 'Government' becomes 'GovXrnmXnt', while a value of 'Enterprise' becomes 'XntXrprise')
`RIGHT`	Extract a substring from a string, starting from the right-most character. The syntax for the `RIGHT` function in Microsoft Excel SQL is: `RIGHT( text, number_of_characters )` where `text` is the string which you wish to extract from, and `number_of_characters` indicates the number of characters that you wish to extract starting from the right-most character.
`RTRIM`	Removes trailing spaces from a string. The syntax for the `RTRIM` function in Microsoft Excel SQL is: `RTRIM( text )` where `text` is the string that you wish to remove trailing spaces from.

Function	Description
`SPACE`	Returns a string value with a specified number of spaces. The syntax for the `SPACE` function in Microsoft Excel SQL is: `SPACE( number )` where `number` is the number of spaces to be returned.
`STR`	Returns a string representation of a number. The syntax for the `STR` function in Microsoft Excel SQL is: `STR( number )` where `number` is the numeric value that you wish to convert to a string.
`TRIM`	Returns a text value with the leading and trailing spaces removed. The syntax for the `TRIM` function in Microsoft Excel SQL is: `TRIM( text )` where `text` is the string that you wish to remove leading and trailing spaces from.
`UCASE`	Converts a string to all uppercase. The syntax for the `UCASE` function in Microsoft Excel SQL is: `UCASE( text )` where `text` is the string which you wish to convert to upper-case.

Resources/Credits

The information in this document was derived from the following sources:

- Writing SQL Queries against Excel files ↗
- MS Excel SQL Query Reference ↗

- Fundamental Microsoft Jet SQL for Access 2000 ↗
- Intermediate Microsoft Jet SQL for Access 2000 ↗
- The Power of SQL Applied to Excel Tables for Fast Results ↗
- Format function ↗
- VBA-(SQL)-String Video-Tutorial for Beginner ↗
- SQL Tutorial for Beginners ↗
- Null Values ↗
- MS Excel: Formulas and Functions ↗

Microsoft provives a free Excel spreadsheet containing sample data ↗ which provides a useful source of Excel data for writing practice queries.

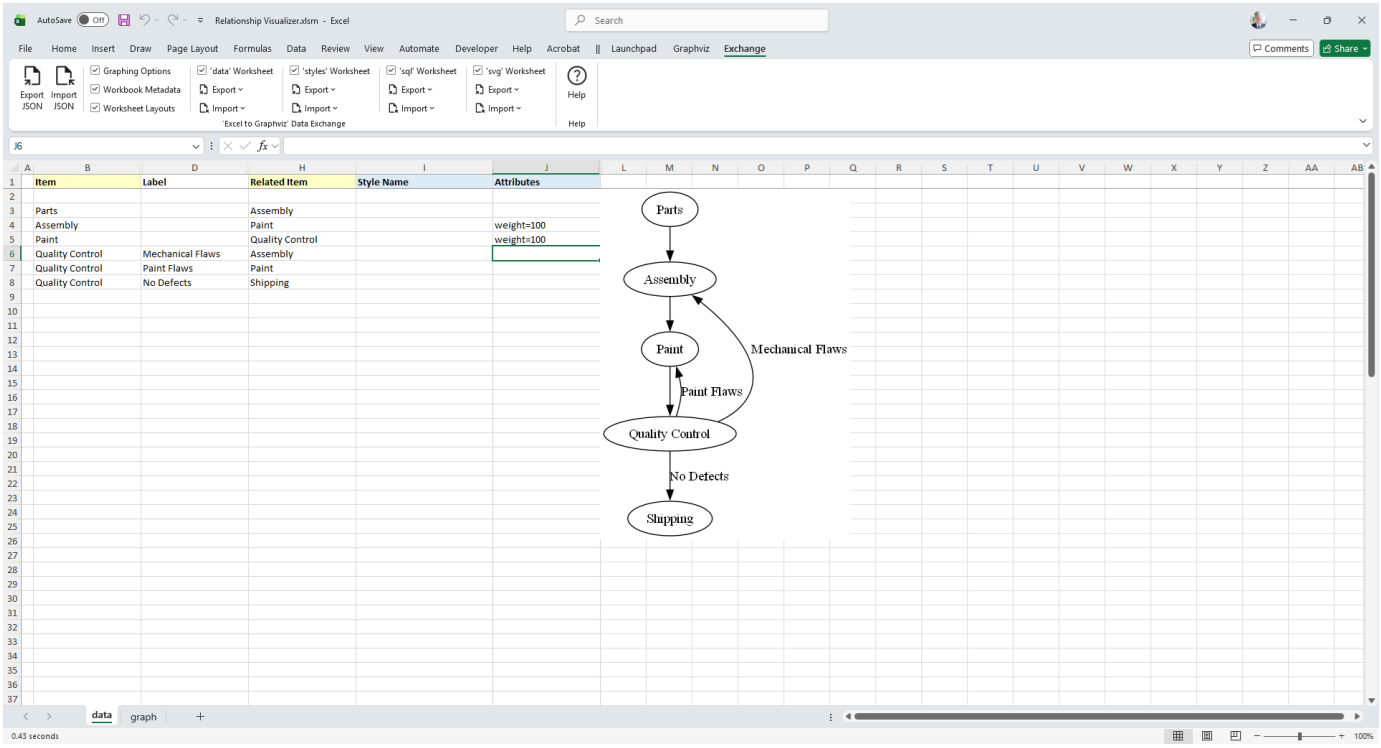
Exchange Data Using JSON Files

There are several drawbacks to using an Excel workbook as your Graphviz IDE:

- **Data and code live in the same file.** As new versions of the workbook are released with additional features, it becomes tedious to copy existing data across multiple worksheets—and to reapply all ribbon settings—from the old version to the new one.
- **Excel workbooks are binary files.** Internally, an Excel file is a ZIP archive. Because it is not a plain text format, it does not work well with version control systems such as Git, nor does it lend itself to meaningful diffs between versions.
- **Text-based sharing services are incompatible.** Platforms such as Pastebin ↗ make it easy to share examples and snippets, but they require text files rather than binary Excel files.
- **Macro-enabled workbooks are often distrusted.** Many users—and many email systems—treat VBA-enabled files as unsafe. It is common for email systems to strip the attachment entirely due to the presence of macros.

These limitations made it clear that a text-based representation of the workbook's data, styles, and settings was needed. The features that support exporting and importing this information are provided on the **Exchange** ribbon tab. There is no associated worksheet, as data exchange operates directly on the internal contents of the workbook.

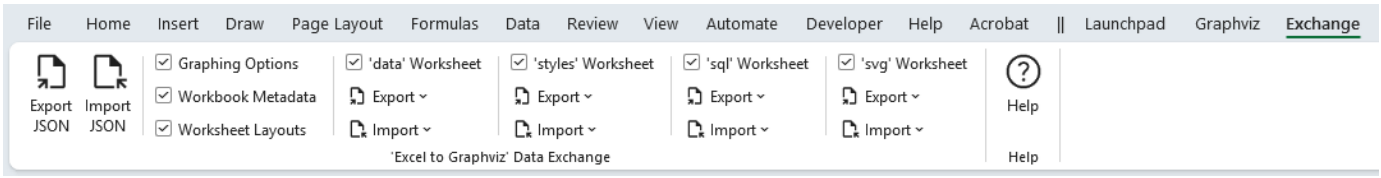
Let's look at an existing spreadsheet and walk through exporting it from one workbook and importing it into another using the Exchange logic. The **Exchange** tab is not associated with a worksheet, and appears as follows:



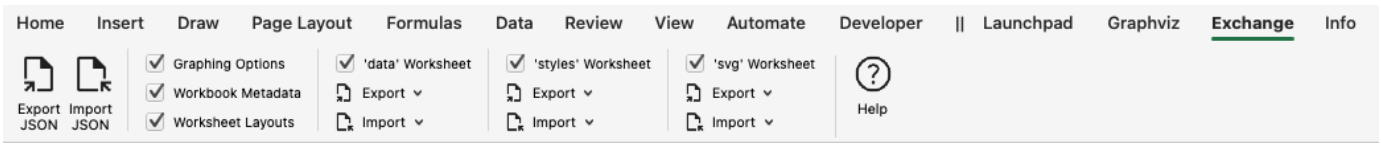
The `Exchange` Ribbon Tab

The `Exchange` ribbon tab appears as follows, and is organized as illustrated

`Windows`



`macOS`



Ribbon Controls

- `Export JSON` - Writes contents out to JSON file

- `Import JSON` - Reads JSON file, and restores data to workbook
- `Graphing Options` - Include options chosen in the ribbons and `settings` worksheet
- `Workbook Metadata` - Include information such as user, Excel version, etc.
- `Worksheet Layouts` - Include information on how the workbook is organized
- `'data' Worksheet` - Include the contents of the `data` worksheet
  - Export
    - `Include row number` - Include the row number of where the data was located
    - `Include row height` - Include the height of the row
    - `Include row visibility` - Include information which tells if the row was visible or hidden
  - Import
    - `Append` - When importing, append the data if existing data exists
    - `Replace` - When importing, ignore any data and replace the contents
- `'styles' Worksheet` - Include the contents of the `styles` worksheet
  - Export
    - `Include row number` - Include the row number of where the data was located
    - `Include row height` - Include the height of the row
    - `Include row visibility` - Include information which tells if the row was visible or hidden
  - Import
    - `Append` - When importing, append the data if existing data exists
    - `Replace` - When importing, ignore any data and replace the contents
- `'sql' - Worksheet` Include the contents of the `sql` worksheet
  - Export
    - `Include row number` - Include the row number of where the data was located
    - `Include row height` - Include the height of the row
    - `Include row visibility` -Include information which tells if the row was visible or hidden
  - Import

    - `Append` - When importing, append the data if existing data exists
    - `Replace` - When importing, ignore any data and replace the contents
- `'svg' - Worksheet` Include the contents of the `svg` worksheet
    - Export
        - `Include row number` - Include the row number of where the data was located
        - `Include row height` - Include the height of the row
        - `Include row visibility` -Include information which tells if the row was visible or hidden
    - Import
        - `Append` - When importing, append the data if existing data exists
        - `Replace` - When importing, ignore any data and replace the contents

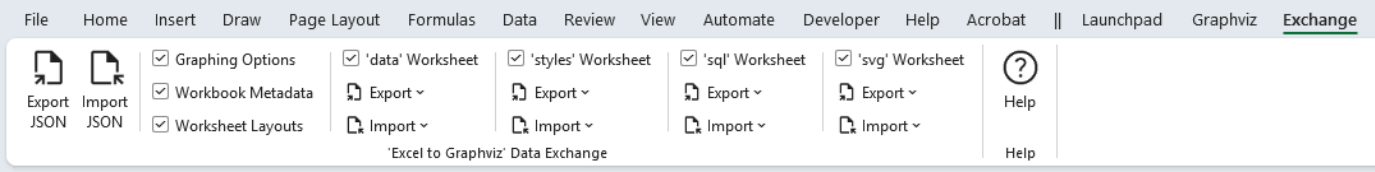
Exporting Data to JSON format

You can export all of the data, or only selected portions, depending on how you intend to use it. Exporting subsets is especially useful when working in teams—for example, you can export just the style definitions to share with others, or export individual team members' data and combine it into a larger workbook using the **Append** option during import.

You may export the entire workbook to a `JSON` file and later import only the sections you need into a new workbook.

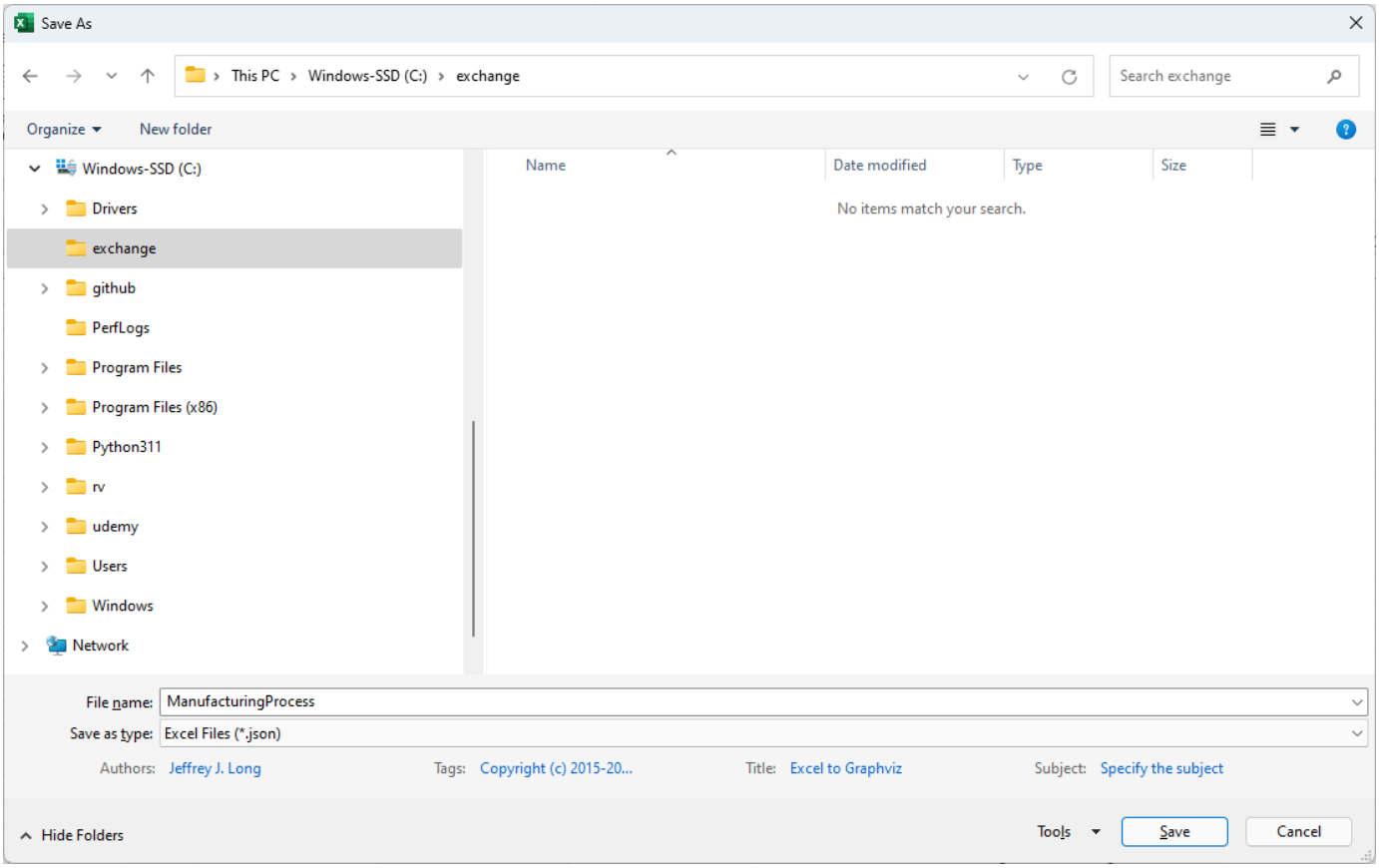
Here are example snippets of exported workbook contents:

Make the selections of the data you wish to export, and press the `Export JSON` button

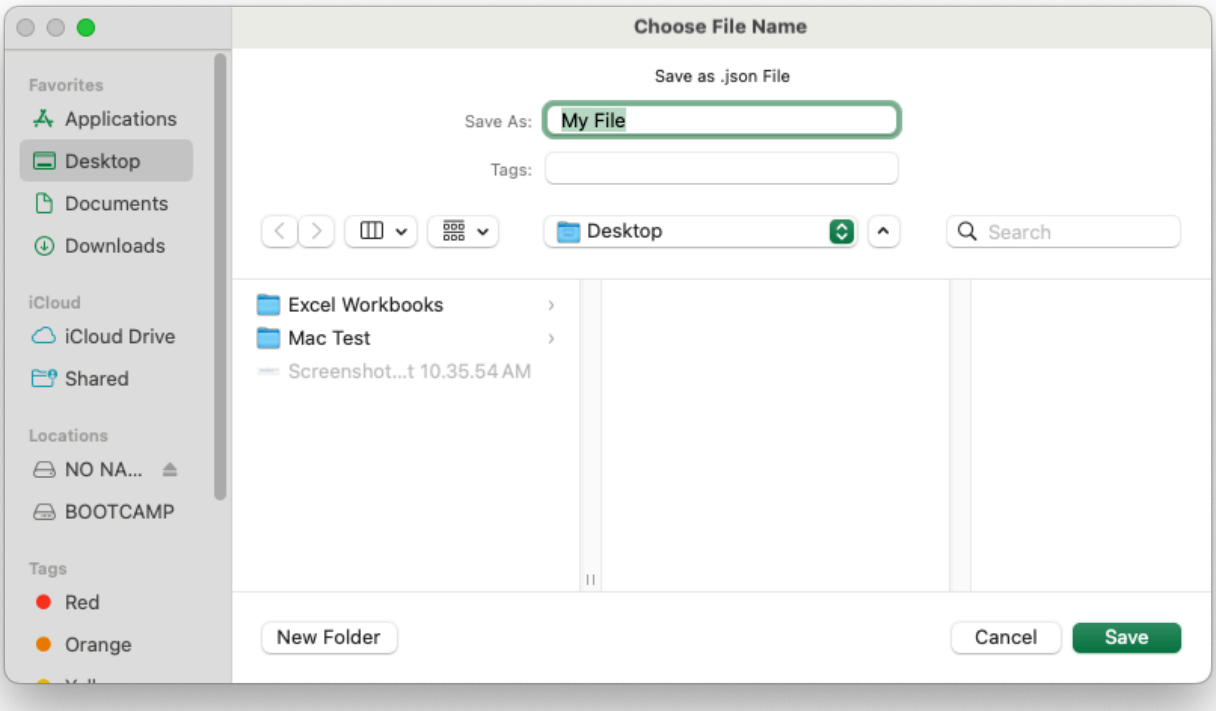


You will be prompted to specify the name of a JSON file that the data should be written to. Enter a file name and press the `Save` button.

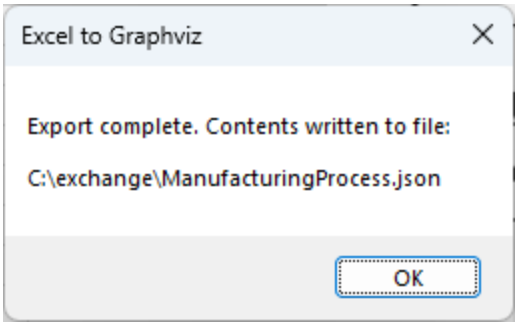
`Windows`


Save As
This PC › Windows-SSD (C:) › exchange
Search exchange
Organize
New folder
Windows-SSD (C:)
Drivers
exchange
github
PerfLogs
Program Files
Program Files (x86)
Python311
rv
udemy
Users
Windows
Network
Name
Date modified
Type
Size
No items match your search.
File name: ManufacturingProcess
Save as type: Excel Files (*.json)
Authors: Jeffrey J. Long
Tags: Copyright (c) 2015-20...
Title: Excel to Graphviz
Subject: Specify the subject
Hide Folders
Tools
Save
Cancel

macOS


Choose File Name
Save as .json File
Save As: My File
Tags:
Desktop
Search
Favorites
Applications
Desktop
Documents
Downloads
iCloud
iCloud Drive
Shared
Locations
NO NA...
BOOTCAMP
Tags
Red
Orange
Excel Workbooks
Mac Test
Screenshot...t 10.35.54 AM
New Folder
Cancel
Save

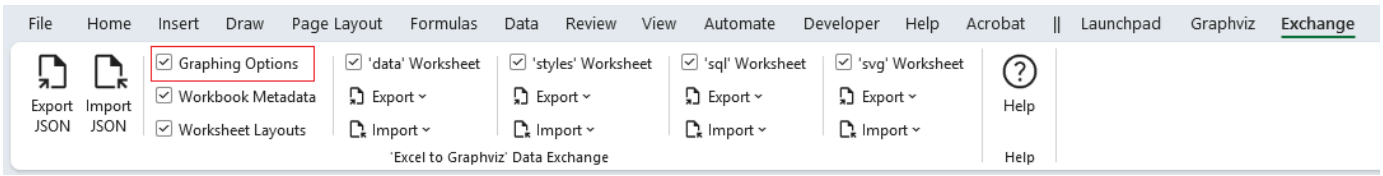
Once the data is written to the file you will receive a pop-up message such as:



Press the **OK** button, and you are done.

The sections which follow provide samples of the JSON exported.

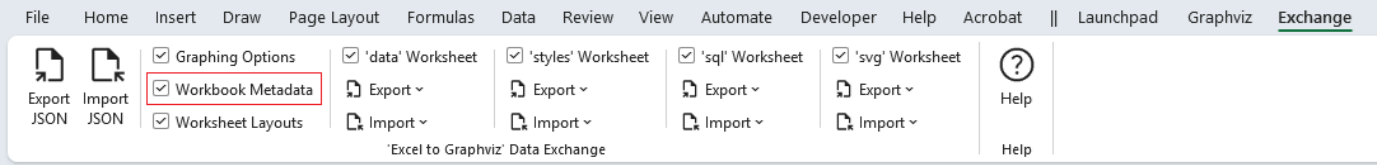
Graphing Options




```json
{
    "settings": {
        "data": {
            "options": {
                "graph": {
                "center": false,
                "clusterRank": "",
                "compound": false,
                "dim": "",
                "dimen": "",
                "forceLabels": false,
                "graphType": "directed",
                "mode": "",
                "model": "",
                "newrank": false,
                "ordering": "",
```

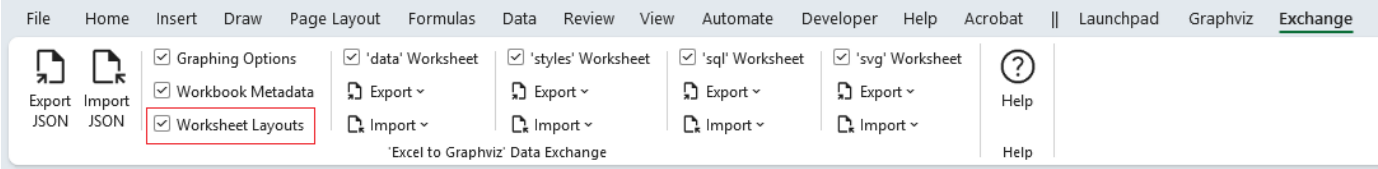
```
            "orientation": false,
            "outputOrder": "",
            "overlap": "",
            "smoothing": "",
            "transparentBackground": false
            },
        --- SNIP ---
```


Workbook Metadata




```json
{
  "metadata": {
    "name": "E2GXF",
    "type": "Excel to Graphviz Exchange Format",
    "version": "1.0",
    "user": "Jeffrey Long",
    "date": "2023-04-09",
    "time": "12:17:35",
    "os": "Windows (64-bit) NT 10.00",
    "excel": "16.0",
    "filename": "Relationship Visualizer.xlsm"
  }
}
```

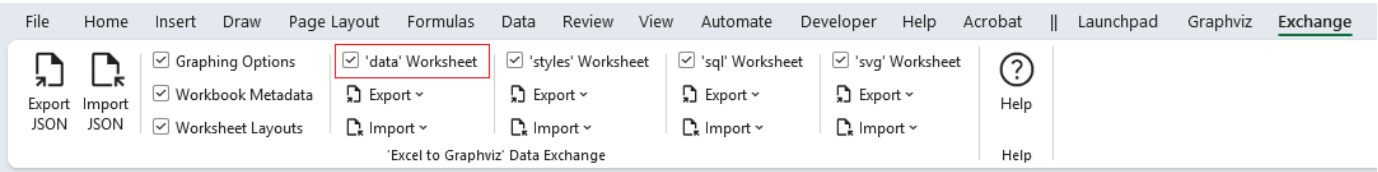

Worksheet Layouts




```json
{
    "layouts": {
        "data": {
            "rows": [
                {
                    "id": "heading",
                    "row": 1,
                    "height": 15,
                    "hidden": false
                },
                {
                    "id": "first",
                    "row": 2,
                    "height": 15,
                    "hidden": false
                }
            ],
            "columns": [
                {
                    "id": "flag",
                    "column": 1,
                    "heading": "",
                    "width": 1.71,
                    "hidden": false,
                    "wrapText": false
                },
                {
                    "id": `Item`,
                    "column": 2,
                    "heading": `Item`,
                    "width": 20.71,
                    "hidden": false,
                    "wrapText": false
                },
```

```
                {
                    "id": "tailLabel",
                    "column": 3,
                    "heading": `Tail Label`,
                    "width": 0,
                    "hidden": true,
                    "wrapText": true
                },
                {
                    "id": `Label`,
                    "column": 4,
                    "heading": `Label`,
                    "width": 0,
                    "hidden": true,
                    "wrapText": true
                }
            ]
        }
    }
    --- SNIP ---
```

'data' Worksheet

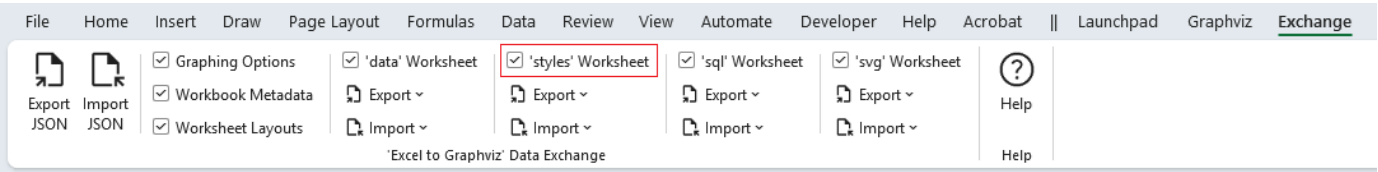



```json
  "content": {
    "data": [
      {
        "row": 2,
        "hidden": false,
        "height": 15
      },
      {
```

```
    "row": 3,
    "hidden": false,
    "height": 15,
    "item": "Parts",
    "relatedItem": "Assembly"
  },
  {
    "row": 4,
    "hidden": false,
    "height": 15,
    "item": "Assembly",
    "relatedItem": "Paint",
    "extraAttributes": {
      "weight": "100"
    }
  },
  {
    "row": 5,
    "hidden": false,
    "height": 15,
    "item": "Paint",
    "relatedItem": "Quality Control",
    "extraAttributes": {
      "weight": "100"
    }
  },
  {
    "row": 6,
    "hidden": false,
    "height": 15,
    "item": "Quality Control",
    "label": "Mechanical Flaws",
    "relatedItem": "Assembly"
  },
  {
    "row": 7,
    "hidden": false,
    "height": 15,
    "item": "Quality Control",
```

```
        "label": "Paint Flaws",
        "relatedItem": "Paint"
      },
      {
        "row": 8,
        "hidden": false,
        "height": 15,
        "item": "Quality Control",
        "label": "No Defects",
        "relatedItem": "Shipping"
      }
    ],
    --- SNIP ---
```

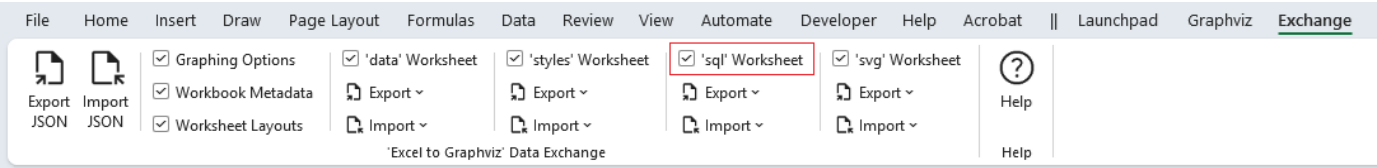

'styles' Worksheet




```json
{
  "content": {
    "styles": [
      {
        "row": 20,
        "hidden": false,
        "height": 45,
        "name": "Border 6 Begin",
        "type": "subgraph-open",
        "format": {
          "penwidth": "1",
          "colorscheme": "reds9",
          "fillcolor": "2",
          "fontname": "Arial Bold",
          "fontsize": "12",
```

```
        "style": "filled",
        "margin": "18"
      },
      "viewSwitches": [
        "yes",
        "no",
        "yes"
      ]
    },
    {
      "row": 21,
      "hidden": false,
      "height": 15,
      "name": "Border 6 End",
      "type": "subgraph-close",
      "viewSwitches": [
        "yes",
        "no",
        "yes"
      ]
    },
    --- SNIP ---
```


'sql' Worksheet

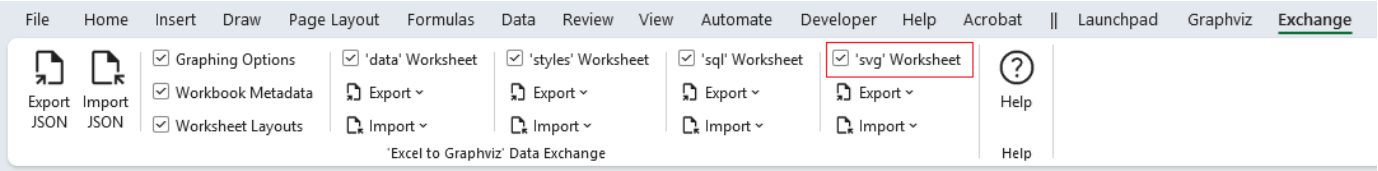



```json
  "content": {
    "sql": [
      {
        "row": 7,
        "hidden": false,
        "height": 310.5,
```

```
            "sqlStatement": "SELECT \u000A  [State Code]                    as [
            "excelFile": "usa states.xlsx",
            "status": "SUCCESS",
            "filters": [
            "EXAMPLE 01",
            "EXAMPLE 01",
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null,
            null
            ]
        },
        {
            "row": 8,
            "hidden": false,
            "height": 15,
            "filters": [
            null,
            null,
            null,
            null,
            null,
            null,
            null,
```

```
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null,
                null
                ]
            },
            --- SNIP ---
```

'svg' Worksheet




```json
{
    "content": {
        "svg": [
            {
                "row": 2,
                "hidden": false,
                "height": 12.75,
                "enabled": false
            },
```

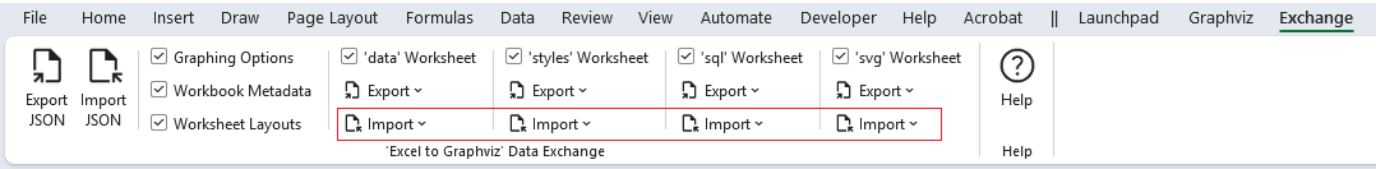
```
        {
            "row": 3,
            "hidden": false,
            "height": 15,
            "enabled": false,
            "find": "Modify the <svg> element to add an onload() function"
        },
        {
            "row": 4,
            "hidden": false,
            "height": 15,
            "enabled": false
        },
        {
            "row": 5,
            "hidden": false,
            "height": 45,
            "find": "xmlns:xlink=\"http://www.w3.org/1999/xlink\">",
            "replace": "xmlns:xlink=\"http://www.w3.org/1999/xlink\" onload=\"makeI
        },
        --- SNIP ---
```

Importing JSON Data

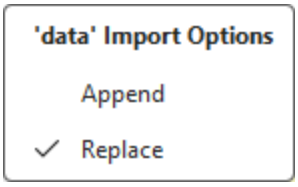
The Import function works the opposite of the Export function. It reads a JSON file which has been exported by the Relationship Visualizer to reconstitute the JSON data into a Relationship Visualizer spreadsheet.

To import a JSON file, start by choosing the sections which you want included. Just as you can export an entire workbook, or sections of the workbook, you may also import an entire workbook or just sections of a workbook.

A key difference for importing worksheets comes via the import option dropdown lists.



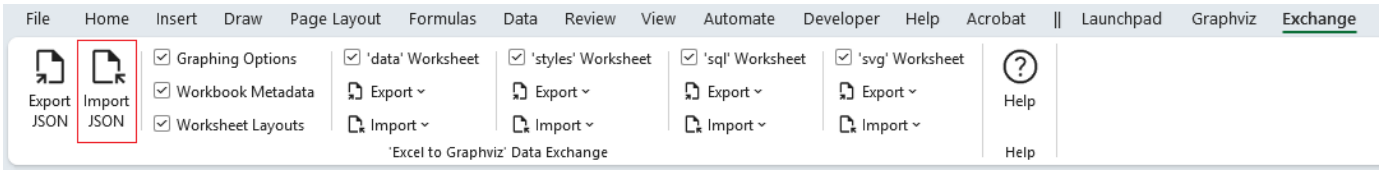
`data`, `styles`, `sql`, and `svg` worksheets choices have Import of Append and Replace.



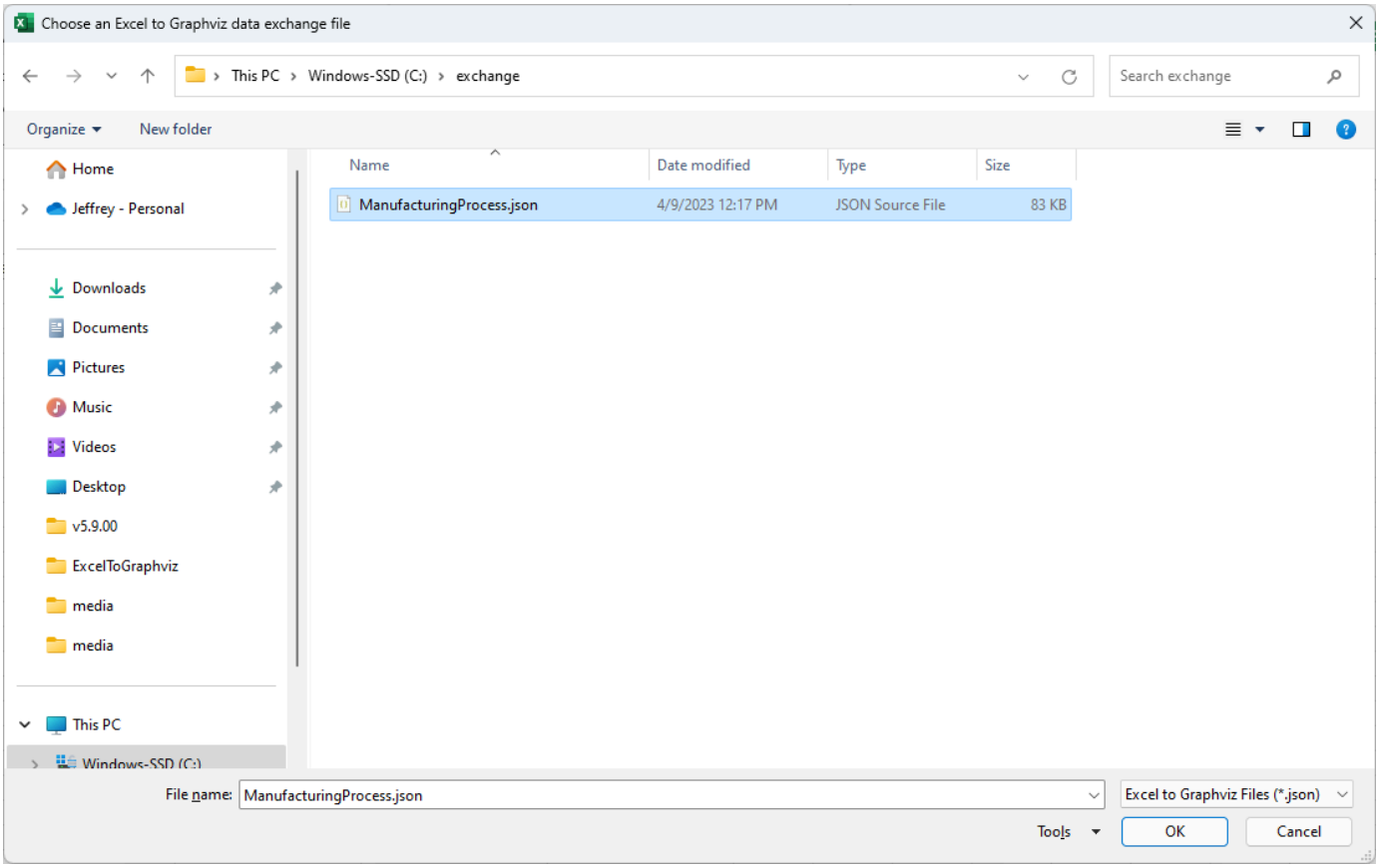
These are mutually exclusive choices that allow you to specify whether to replace the contents in the worksheet, or append the data in the worksheet. **Replace** is the default, and is intended for loading the data into an empty worksheet. **Append** is useful for consolidating data when multiple people are preparing the data.

For example, assume a husband and wife each prepare a family tree of their ancestors. The husband can export his ancestor's data, and the wife can import the husband's data with Append checked. The import will place the data in the `data` worksheet after the wife's data and the family tree will become complete for this family unit.

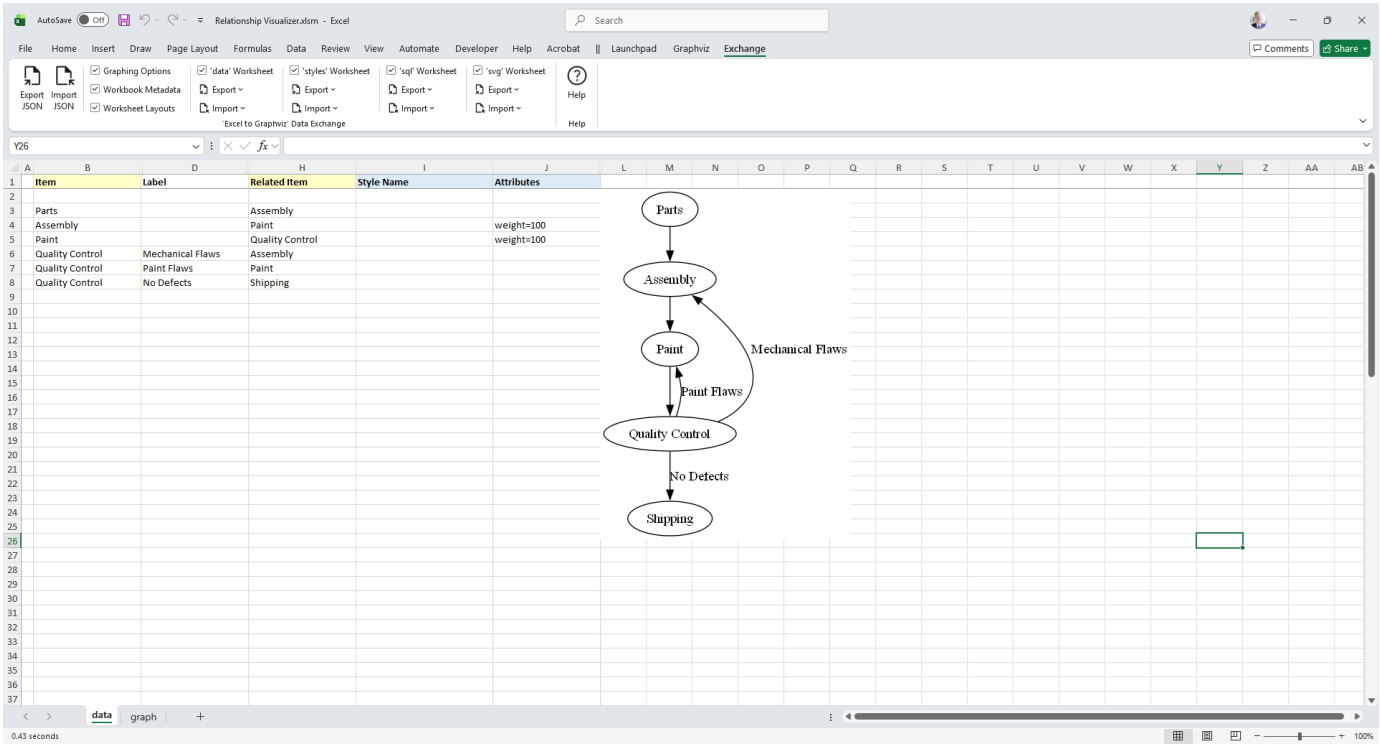
Once you have selected your Import, press the `Import JSON` button.



You will be prompted to **Choose an Excel to Graphviz data exchange file**



Select the file and press the `OK` button. The data will be imported (which may take several seconds). If the `Automatic Refresh` checkbox on the Graphviz tab is checked, the Relationship Visualizer will graph the data to the worksheet, otherwise press the `Refresh Graph` button to see the graph.


Relationship Visualizer.xlsm - Excel
Search
File Home Insert Draw Page Layout Formulas Data Review View Automate Developer Help Acrobat Launchpad Graphviz Exchange
Comments
Share
Export JSON
Import JSON
Graphing Options
Workbook Metadata
Worksheet Layouts
'data' Worksheet
Export
Import
'styles' Worksheet
Export
Import
'sql' Worksheet
Export
Import
'svg' Worksheet
Export
Import
Excel to Graphviz
Data Exchange
Help
Help
Y26
Item
Label
Related Item
Style Name
Attributes
Parts
Assembly
Assembly
Paint
weight=100
Paint
Quality Control
weight=100
Quality Control
Mechanical Flaws
Assembly
Quality Control
Paint Flaws
Paint
Quality Control
No Defects
Shipping
Parts
Assembly
Paint
Mechanical Flaws
Paint Flaws
Quality Control
No Defects
Shipping
data
graph
0.43 seconds
100%

Viewing DOT Source Code

The Relationship Visualizer is designed to let you create Graphviz graphs without needing to learn the DOT Language ↗. However, it can also serve as an effective learning tool for DOT itself. This topic explains how to access the DOT source code generated from the Excel worksheets and passed to the Graphviz layout engine. These capabilities allow you to build relationship graphs and then examine the underlying code that produces them.

Why is this useful?

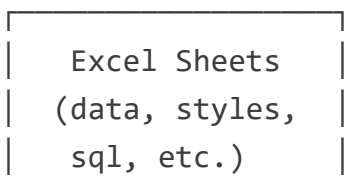
Understanding how the Relationship Visualizer transforms worksheet data into DOT source code can help you:

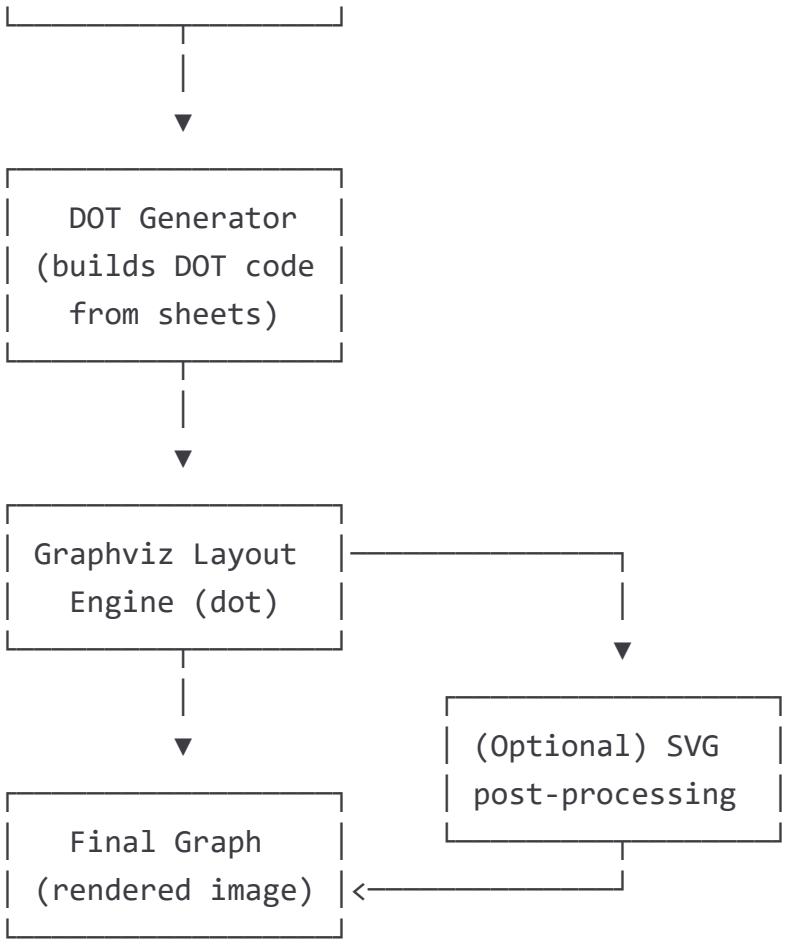
- **Learn the DOT language naturally** by seeing real examples generated from your own graphs
- **Validate the structure** of the code being sent to the Graphviz layout engine
- **Troubleshoot layout issues** by comparing the visual output with the underlying DOT
- **Experiment with advanced features** by editing or extending the generated code
- **Build confidence** in how your relationships, attributes, and styles are translated into Graphviz syntax

This makes the tool not only a no-code graph builder, but also a gentle, hands-on learning environment for anyone curious about DOT.

Graph Creation Steps

The diagram below shows the flow from your worksheet data to the final Graphviz output. It highlights how the Relationship Visualizer generates DOT source and passes it to the Graphviz layout engine.

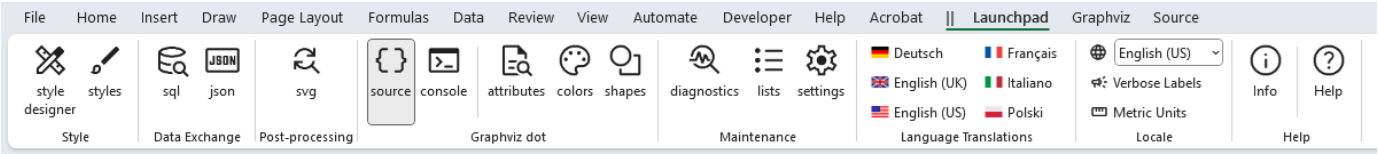




The `source` Worksheet

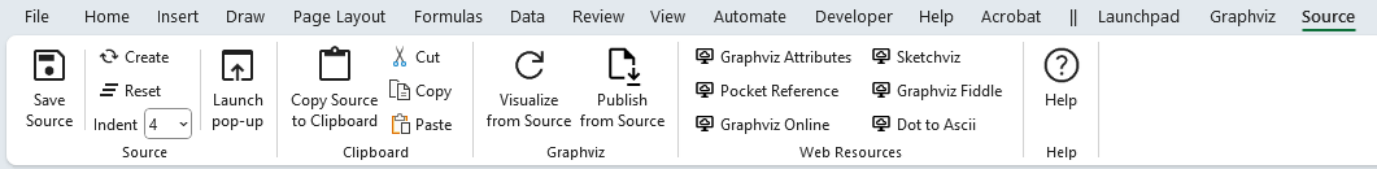
The Relationship Visualizer includes a worksheet named `source`, which displays the DOT source code generated each time a graphing button is pressed.

The `source` worksheet is hidden by default. To reveal it, select the **source** button in the **Graphviz dot** section of the `Launchpad` tab.



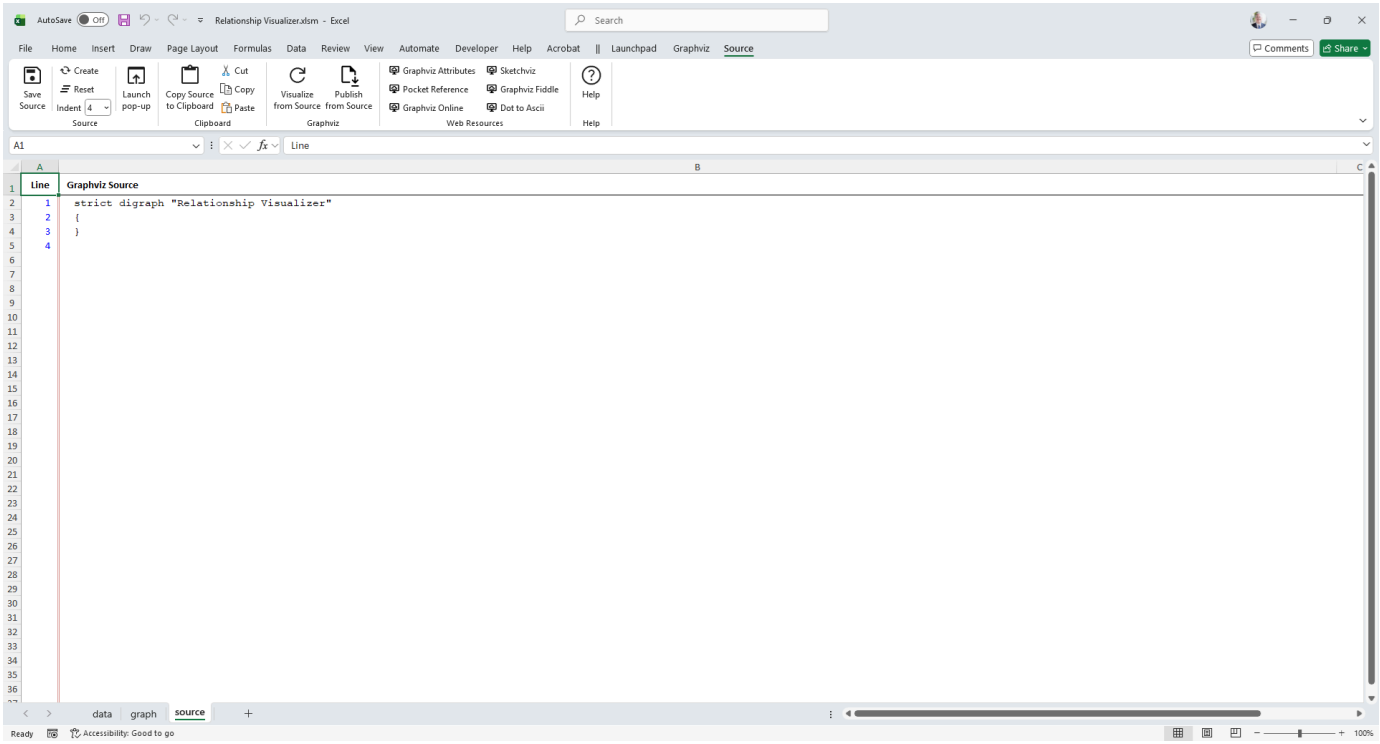
Click the `source` worksheet to make it the active sheet. When `source` is active, the `Source` ribbon tab is automatically displayed. It appears as follows:

`Windows`

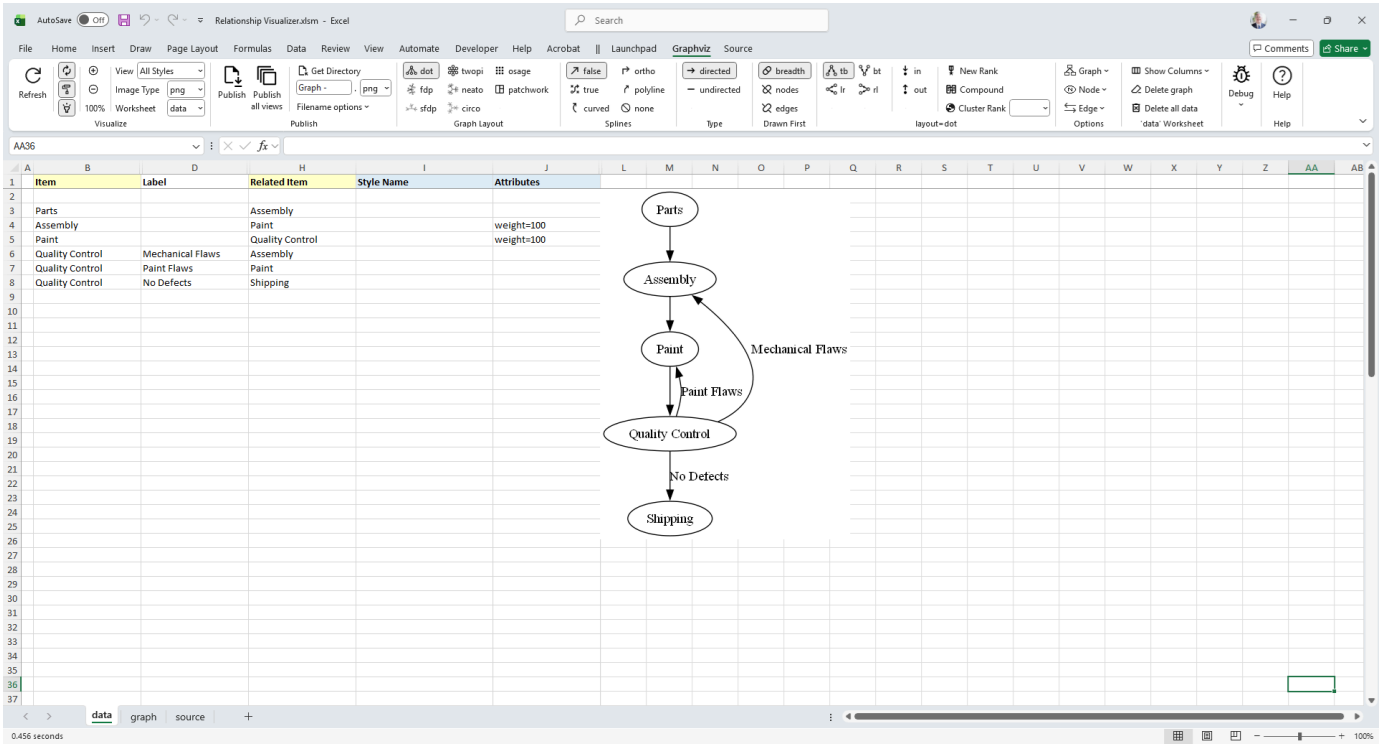


`macOS`

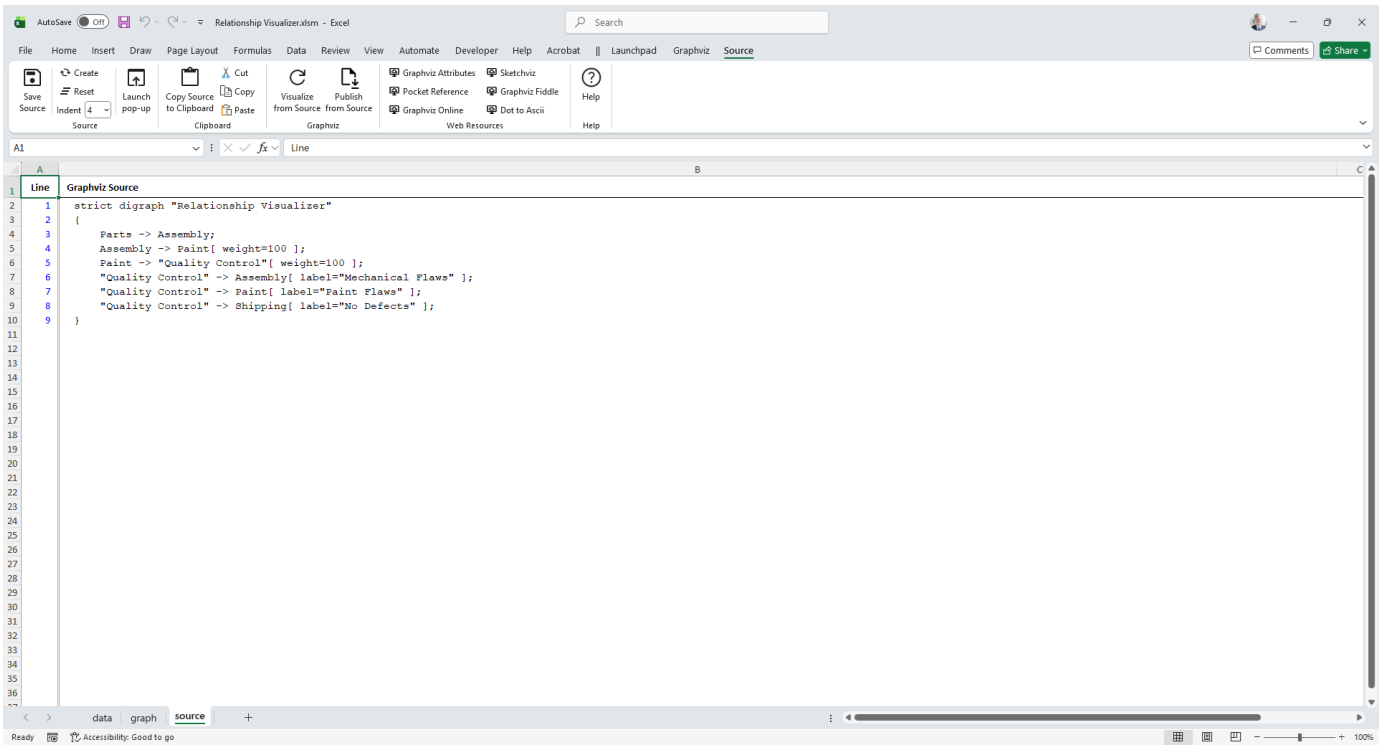
The next image shows the default DOT code when the `data` worksheet does not contain any content. As you create a graph the source worksheet will update.



Switch to the `data` worksheet and build a graph. Consider the following example:

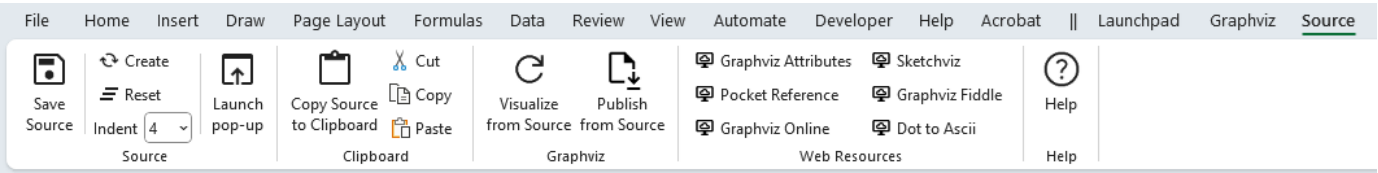


When we switch back to the `source` worksheet the Graphviz source code appears as:

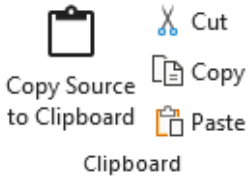
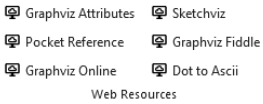


The `Source` Ribbon Tab

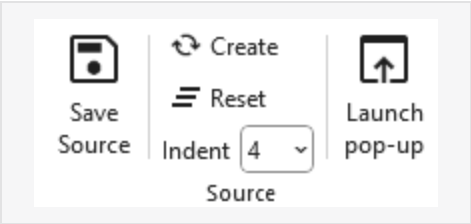
Now that you understand the basics of viewing Graphviz source code, let us look at the features contained in the `Source` ribbon tab. The `Source` ribbon tab is activated whenever the `source` worksheet is activated. It appears as follows:



It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
Source		Controls for viewing and saving `DOT` source code.
Clipboard		Controls for copying the `DOT` source to the clipboard.
Graphviz		Generates a graph from the Graphviz data on the `source` worksheet using the settings on the `Graphviz` ribbon tab.
Web Resources		Six buttons which can launch the user's default browser and display a web page pertaining to a Graphviz topic.
Help		Provides a link to the `Help` content for the `Source` worksheet (i.e. this web page).

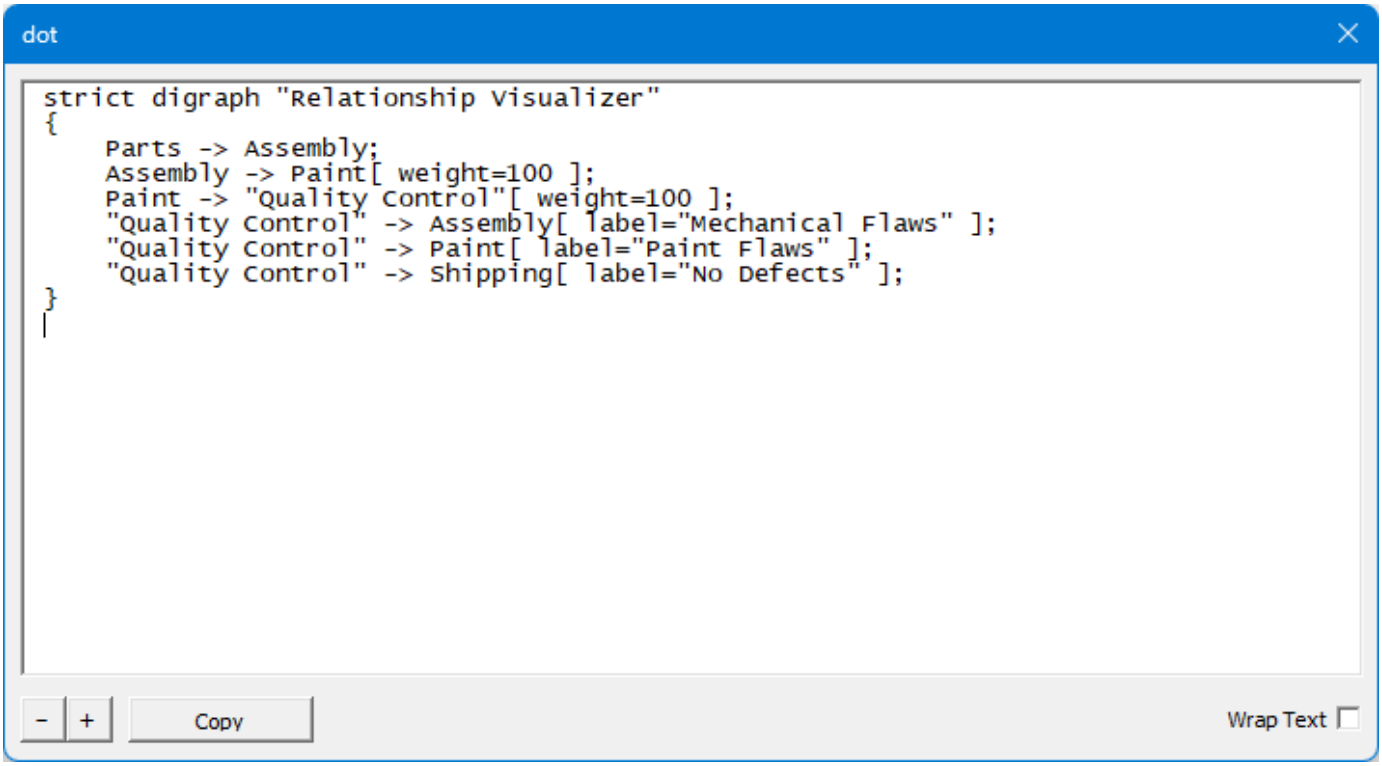
Source



Controls for viewing and saving `DOT` source code.

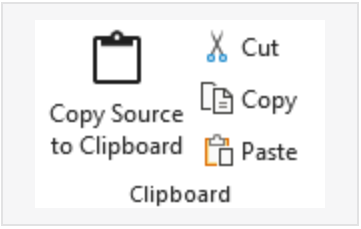
Label	Control Type	Description
Save Source	Button	Saves the Graphviz source code displayed on the `source` worksheet to a file.
Create	Button	Generates fresh Graphviz source code from the data worksheet without invoking Graphviz to render a graph. This is useful when you have manually edited the DOT source and want to restore it to its original, workbook-generated form.
Reset	Button	Clears all data on the source worksheet, but leaves the headings.
Indent	Dropdown List	Number of spaces equaling a tab indentation
Launch pop-up	Button	Opens a pop-up window that displays the DOT source code. This window is set to “always on top,” allowing you to switch worksheets while still viewing the source. It is especially useful for observing how the DOT code updates as you modify the data and the graph evolves.

The pop-up version of the DOT source code appears as follows:



The form is primarily read-only, however you can enlarge or shrink the font size, copy the contents to the clipboard, and word-wrap long text.

Clipboard



Controls for copying the `DOT` source to the clipboard. This capability is only present on Windows, as the Windows API code it relies on is not present on macOS.

Label	Control Type	Description
Copy Source to Clipboard	Button	Selects all the Graphviz source code and copies it to the clipboard.
Cut	Button	Standard Excel cell-based `Cut`

Label	Control Type	Description
Copy	Button	Standard Excel cell-based `Copy`
Paste	Button	Standard Excel cell-based `Paste`

Graphviz

Utilility to visualize the source code as a Graphvis graph. T

Label	Control Type	Description
Visualize from Source	Button	Generates a graph from the Graphviz data on the `source` worksheet using the settings on the `Graphviz` ribbon tab, and displays the result on the `graph` worksheet. Note that this data flow is one-way: changes made on the `data` worksheet can be regenerated as `source`, but edits made directly on the `source` worksheet are not detected and will not flow back into the `data` worksheet.
Publish from Source	Button	Graphs the Graphviz data on the source worksheet using the settings on the `Graphviz` ribbon tab and writes the graph to a file. All the restrictions noted for the `Refresh Graph` button apply to this action as well.

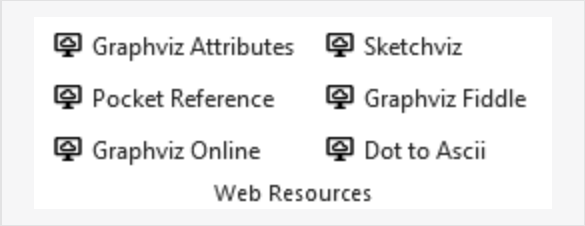
TIP

You can modify the DOT source code on this worksheet, and update the graph using the visualize buttons.

WARNING

Changes to the DOT source code are not reflected on the `data` worksheet. Any changes made on the `data` worksheet will wipe out any changes you make on the `source` worksheet.

Web Resources



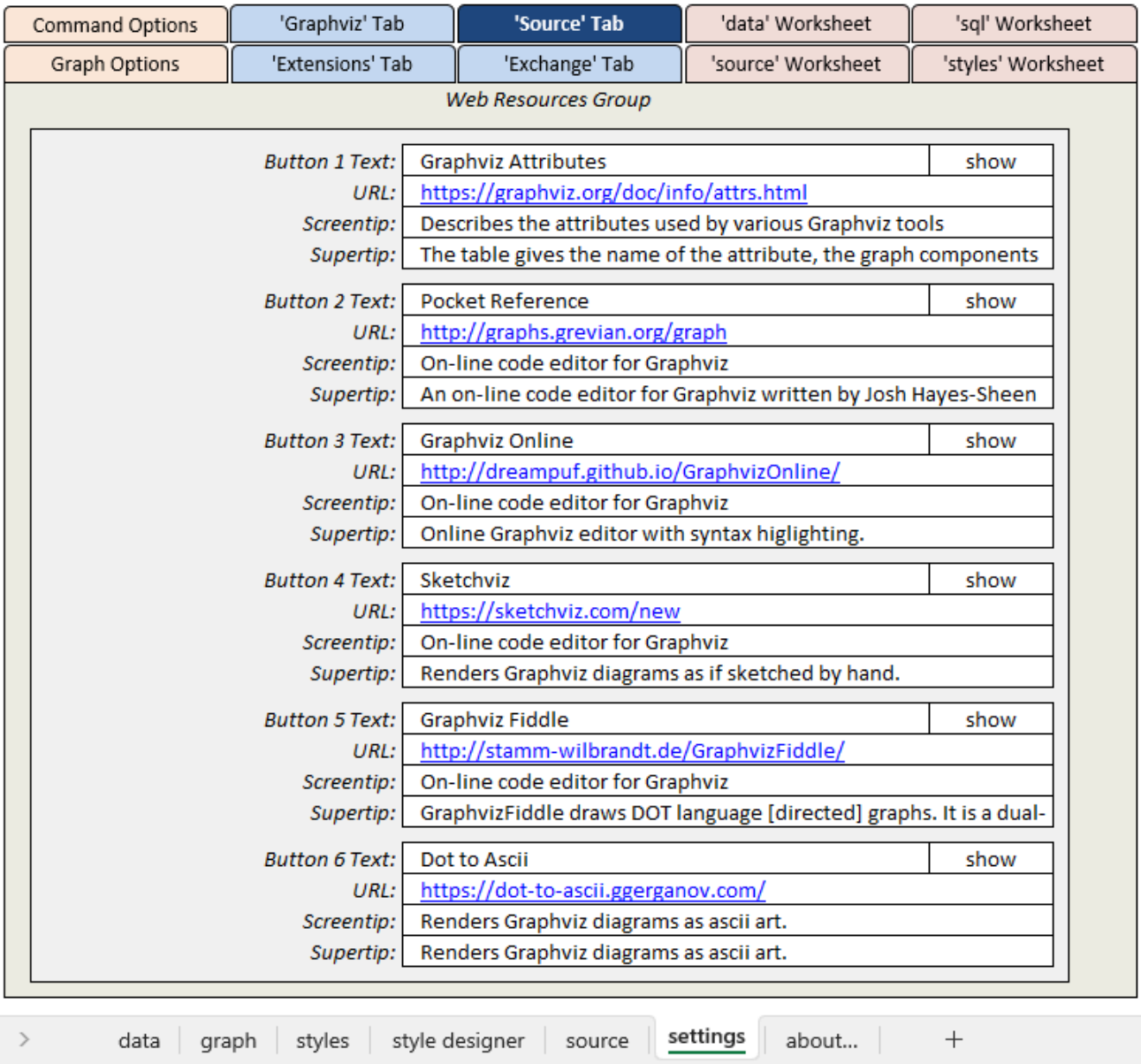
The `Web Resources` group dynamically supports six buttons which can launch the user's default browser and display a web page. Button text, tool tips, and the URL are specified in the `settings` worksheet. The worksheet as distributed contains links to Graphviz documentation and online Graphviz rendering tools.

The default settings are:

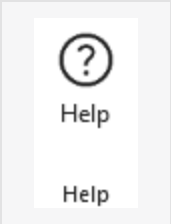
#	Label	Control Type	Description
1	Graphviz Attributes ↗	Button	Describes the attributes used by various Graphviz tools.
2	Pocket Reference ↗	Button	An on-line code editor for Graphviz written by Josh Hayes-Sheen for Computer Science students.
3	Graphviz Online ↗	Button	Online Graphviz editor with syntax highlighting.
4	Sketchviz ↗	Button	Online Graphviz editor which creates the graph as if it were sketched by hand.
5	Graphviz Fiddle ↗	Button	Graphviz Fiddle draws DOT language [directed] graphs. It is a dual-view Graphviz editor and playground for Graphviz written by Hermann Stamm-Wilbrandt.

#	Label	Control Type	Description
6	Dot to Ascii ↗	Button	Renders Graphviz diagrams as ascii art.

The values associated with these buttons can be changed to suit your own preferences. These links are located in the `settings` worksheet in the `Ribbon Options` location, in the `Source` tab



Help



Provides a link to the `Help` content for the `Source` worksheet (i.e. this web page).

Label	Control Type	Description
Help	Button	Provides a link to this web page.

Interactive Graphviz over the Internet

There are many public web sites which will allow you to interactively edit DOT code in a browser window and see the corresponding graph.

Sites you can explore include:

- Pocket Reference ↗
- Graphviz Online ↗
- Sketchviz ↗
- Graphviz Fiddle ↗
- Dot to Ascii ↗
- Graphviz Visual Editor ↗

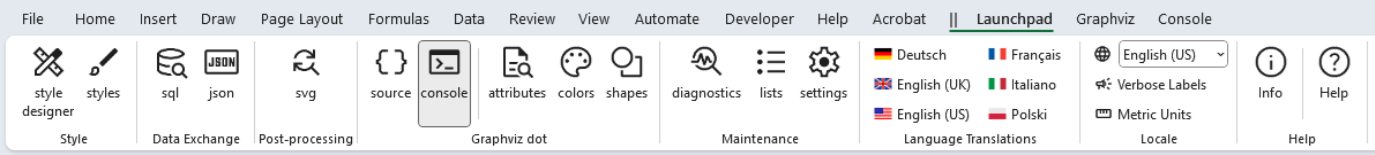
These sites work best with small graphs and cannot handle features like displaying images. There is no guarantee that these sites will continue to operate into the future. They do, however, make it easy to edit DOT graphs, quickly see the results, and learn the DOT programming language without having to install Graphviz.

DOT Message Console

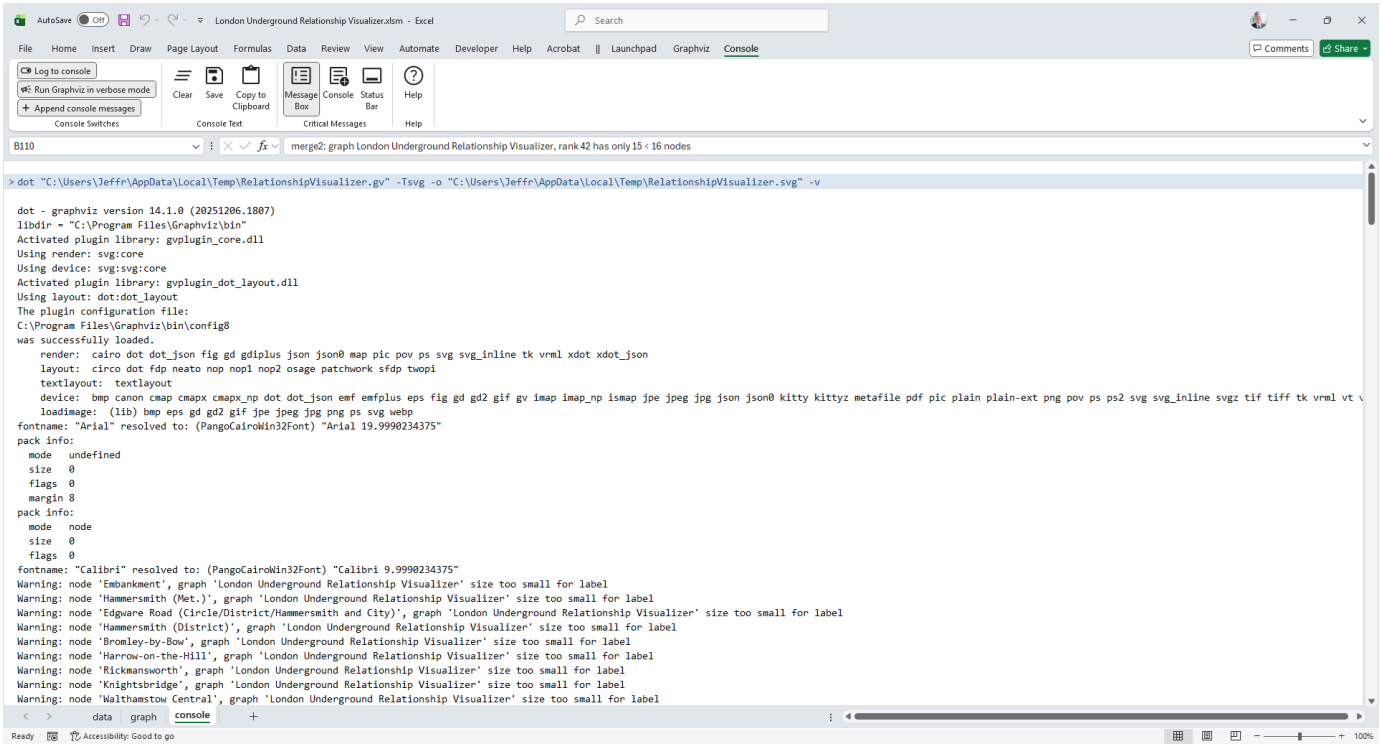
The `console` worksheet shows the messages emitted by the `dot` command when Graphviz runs.

The `console` Worksheet

The `console` worksheet is reached from the `Graphviz dot` section of the Launchpad ribbon tab.



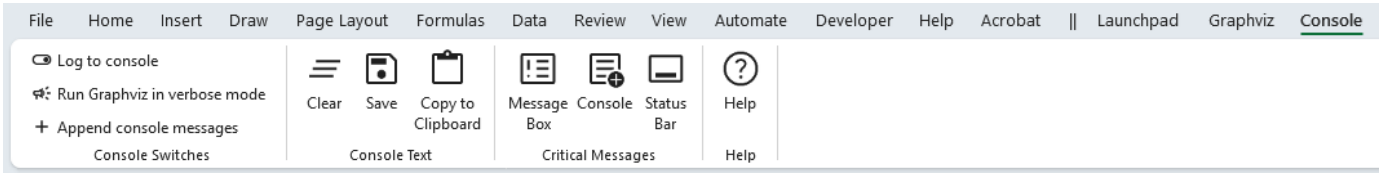
Messages in the shaded rows are the `dot` commands issued when a Graphviz visualization is requested using the `dot` command. The messages emitted by `dot` are displayed against a white background. You have the choice of standard or verbose messages. You can also restrict the messages to the most recent dot invocation or append them in a running log.



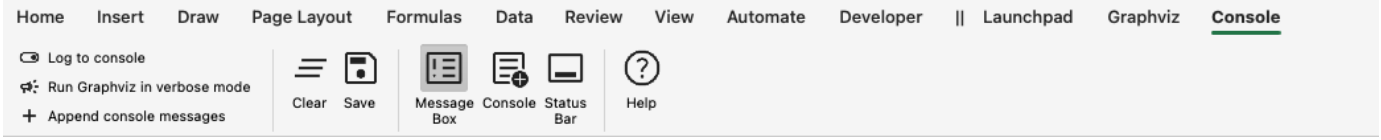
The `Console` Ribbon Tab

The **Console** ribbon tab is activated whenever the *console* worksheet is opened from the Launchpad. It appears as follows:

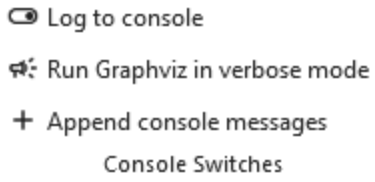
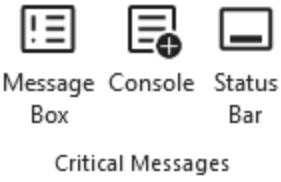
`Windows`



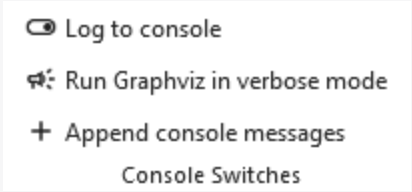
`macOS`



It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls
Console Switches	
Console Text	
Critical Messages	
Help	

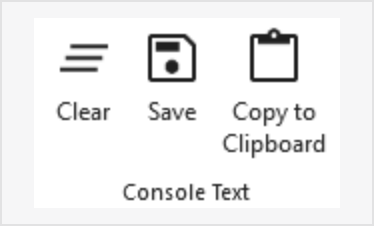
Console Switches



Label	Control Type	Description
Log to console	Checkbox	Turns logging of the Graphviz `dot` messages on/off.
Run Graphviz in verbose mode	Checkbox	When checked, appends the `-V` flag to the `dot` command which tells Graphviz to emit verbose

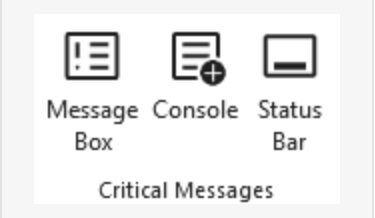
Label	Control Type	Description
		messages.
Append console messages	Checkbox	Allows the console to display messages from a single `dot` invocation, or maintain a running log.

Console Text



Name	Control Type	Description
Clear	Button	Resets the contents of the `console` worksheet.
Save to File	Button	Brings up a file save dialog which lets you save the console messages to a file.
Copy to Clipboard	Button	Copies the contents of the `console` worksheet to the clipboard (Windows OS only).

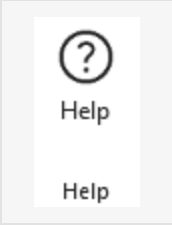
Critical Messages



Name	Control Type	Description
Log to console	Toggle Button	Enables or disables logging of Graphviz `dot` messages to the console.
Run Graphviz in verbose mode	Toggle Button	Appends the `-V` flag to the `dot` command, causing Graphviz to emit verbose diagnostic messages.
Append console messages	Toggle Button	Determines whether the console shows only the current `dot` invocation or maintains a running log.

These buttons operate independently. You can deselect all of them to run silently, with no errors reported; select all of them so every reporting channel is used; or choose any combination in between.

Help



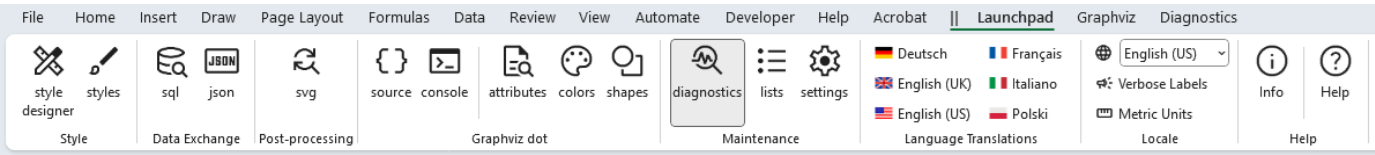
Provides a link to the `Help` content for the `console` worksheet (i.e. this web page).

Name	Control Type	Description
Help	Button	Provides a link to this web page.

Diagnostics Worksheet

The `diagnostics` Worksheet

The `diagnostics` worksheet is reached from the `Maintenance` section of the Launchpad ribbon tab.

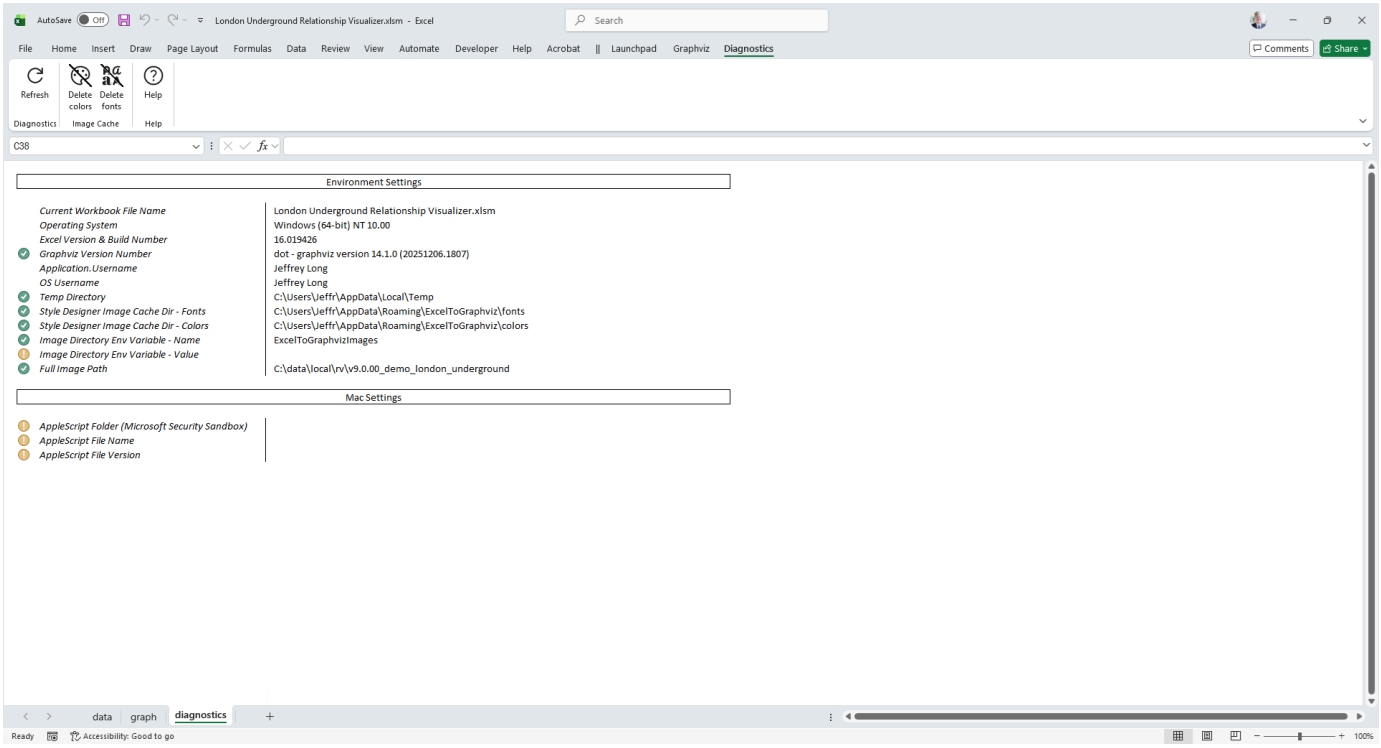


The `diagnostics` worksheet key information about the workbook and the environment in which it is being run.

The information displayed on the `diagnostics` worksheet includes:

- The workbook file name
- The Operating System (Windows 64-bit, Windows 32-bit, Mac)
- The version of Graphviz installed
- Directories for temporary files
- Directories containing cached images displayed by the `style designer`

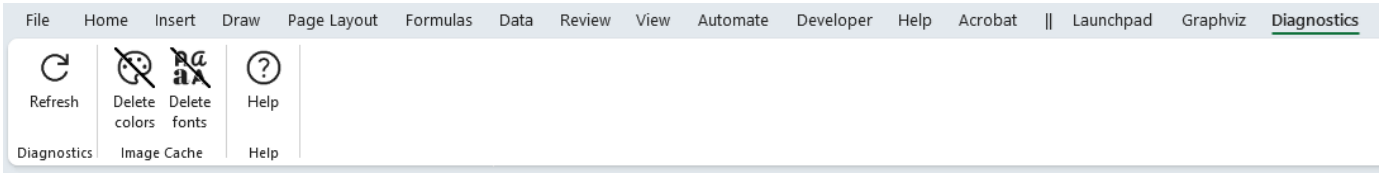
The `diagnostics` worksheet appears as follows (on Windows):



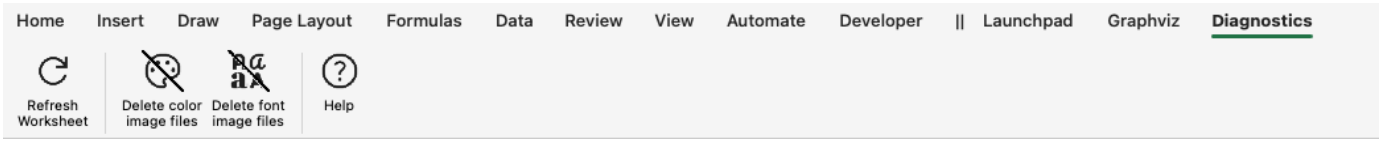
The `Diagnostics` Ribbon Tab

The `Diagnostics` ribbon tab is activated whenever the `diagnostics` worksheet is activated from the Launchpad. It appears as follows:

`Windows`



`macOS`



It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
Diagnostics		Refreshes key information about the workbook and the environment in which it is being run.
Image Cache		Provides an easy way to delete the images created by the **Style Designer** which are preview thumbnails for fonts and colors.
Help		Provides a link to the `Help` content for the `diagnostics` worksheet (i.e. this web page).

Diagnostics

Refreshes the diagnostics values displayed. Useful if you have changed something on your system such as installing a different version of Graphviz, and you want to confirm what is being used.

Name	Control Type	Description
Refresh	Button	Forces a refresh of the information on the page to ensure no data was carried over from a workbook saved elsewhere.

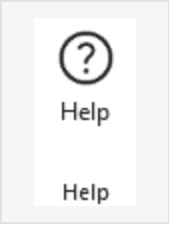
Image Cache

Provides an easy way to delete the images created by the **Style Designer** which are preview thumbnails for fonts and colors.

Name	Control Type	Description
Delete colors	Button	Deletes the images used to display color swatches next to the color names.
Delete fonts	Button	Deletes the font preview images generated by Graphviz to show how each font is rendered.

Note that image files are cached in memory if you visit the *Style Designer* worksheet before pressing these buttons. These controls delete only the files on disk. To force new preview images to be generated, close and reopen the workbook.

Help



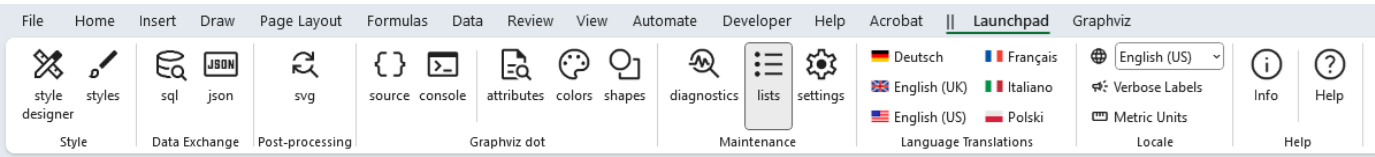
Provides a link to the `Help` content for the `diagnostics` worksheet (i.e. this web page).

Name	Control Type	Description
Help	Button	Provides a link to this web page.

Lists Worksheet

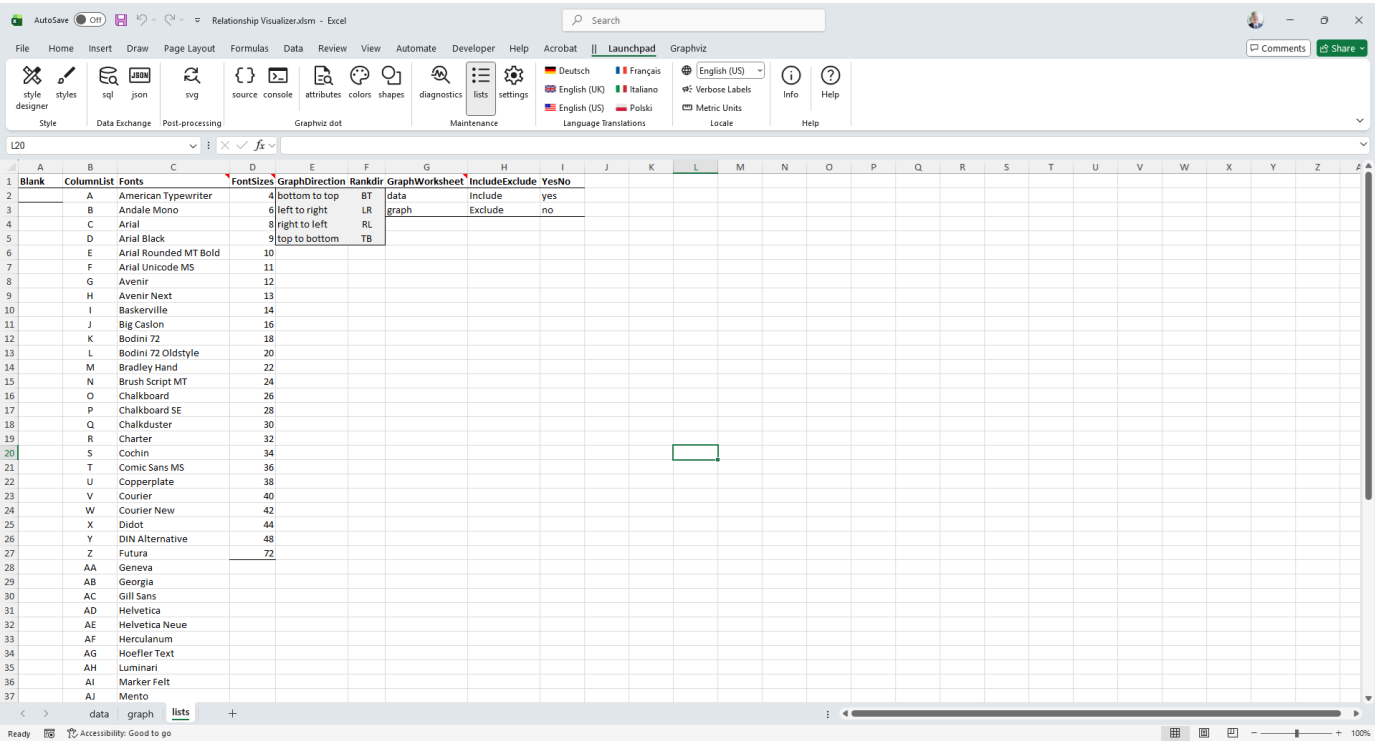
The `lists` Worksheet

The `lists` worksheet is reached from the `Maintenance` section of the Launchpad ribbon tab.



The `lists` worksheet maintains a set of lists used by cell validations on the `settings` worksheet and by the dropdown controls in the `style designer` ribbon tab.

The `lists` worksheet appears as follows:



Blank	ColumnList	Fonts	FontSizes	GraphDirection	Rankdir	GraphWorksheet	IncludeExclude	YesNo
	A	American Typewriter	4	bottom to top	BT	data	Include	yes
	B	Andale Mono	6	left to right	LR	graph	Exclude	no
	C	Arial	8	right to left	RL			
	D	Arial Black	9	top to bottom	TB			
	E	Arial Rounded MT Bold	10					
	F	Arial Unicode MS	11					
	G	Avenir	12					
	H	Avenir Next	13					
	I	Baskerville	14					
	J	Big Caslon	16					
	K	Bodini 72	18					
	L	Bodini 72 Oldstyle	20					
	M	Bradley Hand	22					
	N	Brush Script MT	24					
	O	Chalkboard	26					
	P	Chalkboard SE	28					
	Q	Chalkduster	30					
	R	Charter	32					
	S	Cochin	34					
	T	Comic Sans MS	36					
	U	Copperplate	38					
	V	Courier	40					
	W	Courier New	42					
	X	Didot	44					
	Y	DIN Alternative	48					
	Z	Futura	72					
	AA	Geneva						
	AB	Georgia						
	AC	Gill Sans						
	AD	Helvetica						
	AE	Helvetica Neue						
	AF	Herculanum						
	AG	Hoefler Text						
	AH	Luminari						
	AI	Marker Felt						
	AJ	Mento						

The **Fonts** list is primarily relevant for macOS users. On Windows, the Relationship Visualizer can retrieve the installed fonts automatically, but macOS does not provide a way

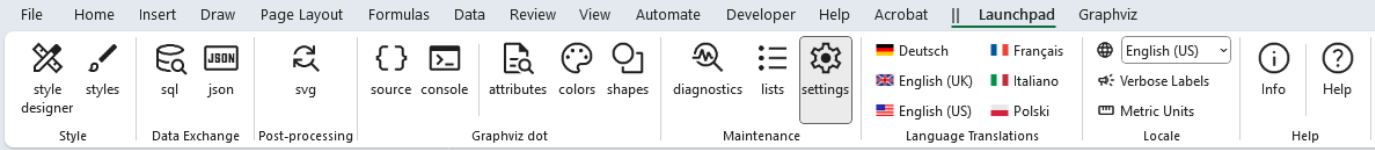
to obtain this list programmatically. As a result, the font list shown on macOS comes from the `Fonts` list in column C.

You may add or remove font names in this list to match the fonts available on your Mac. After making changes, you must close and reopen the workbook for the updated list to appear.

Changing Master Settings

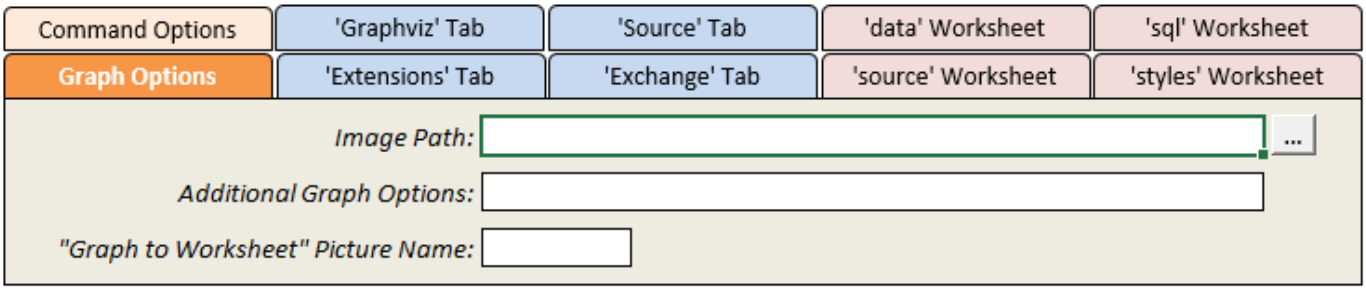
The `settings` worksheet is where you can customize overarching settings which control how the graph is created, specify directories containing images to use in the graph, control how long the graphing engine is allowed to run, specify Graphviz command line parameters, and modify the order in which the worksheet columns are laid out.

The `settings` worksheet is displayed by selecting the **Settings** button on the `Launchpad` ribbon tab. This sheet provides access to the configuration values that control how the Relationship Visualizer operates.



The following sections provide a brief description of the settings:

Graph Options



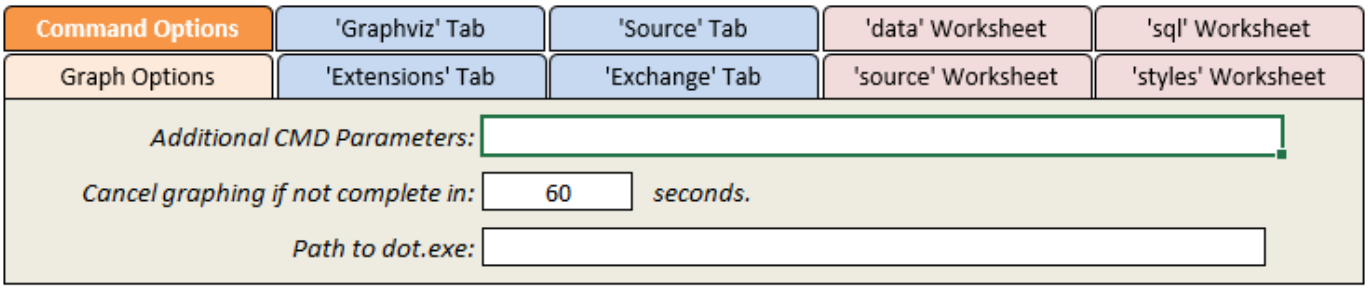
These settings control the graph options used to produce the diagrams.

Setting	Description
Image Path	Specifies one or more directories when images to include in the graph are stored. The Relationship Visualizer looks for images in the same directory where the spreadsheet is stored first, followed by the directory specified here. The folder picker allows you to select one directory, however multiple

Setting	Description
	directories can be specified if the directories are separated by a semi-colon `;`. If you use the `style designer` worksheet to design an image node the directory containing the image will be automatically appended to the value here, unless the directory is already listed or is the directory where the Excel spreadsheet is stored.
Additional Graph Options	A means to insert additional Graphviz graph directives into the graph at the top-most level. To use this setting, you must understand how Graphviz works. This setting can be used to control aspects such as node spacing. For example: `sep="+30,300"`
"Graph to Worksheet" Picture Name	The name to assign to the picture object containing the bitmap which is created by the `Refresh Graph` button on the `Data` worksheet. If this value is blank, Excel assigns a name.

Command Options

The `Command Options` settings allow you to pass parameters to the Graphviz command line programs.



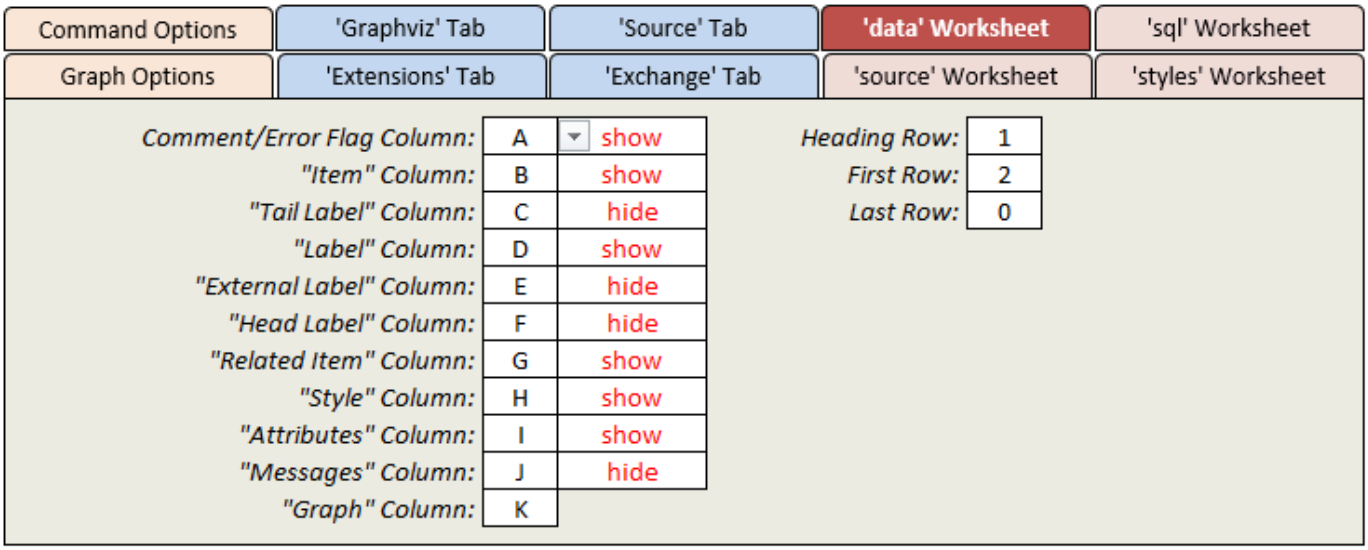
Setting	Description
Additional CMD Parameters	A means to pass additional command line parameters to the layout engine program. For example: -Efontsize="8" -Efontname="Arial" would override

Setting	Description
	the default edge format. Setting the font size to 8 points, and the font to Arial for all edges which do not have explicit style specifications.
Cancel graphing if not complete in __ seconds	The maximum number of seconds the graphing engine can run. If set to 0, no time limit exists.
Path to dot.exe	The path to the dot executable if Graphviz is not installed/not available via the PATH environment variable.

Worksheet Settings

'data' Worksheet

These settings allow you to rearrange the column layout of the data worksheet, and restrict which rows get processed.

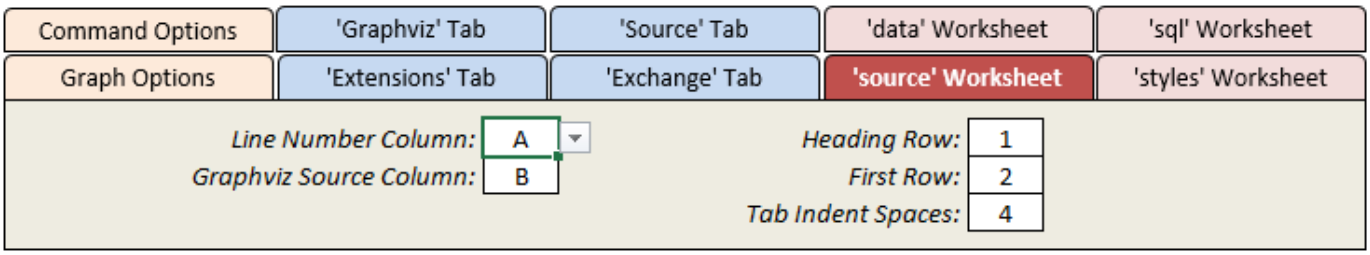


Setting	Description
`Comment/Error Flag` Column	The column used to place '#' comment characters which indicate that the rows should be treated as a comment and not be included in the data to be graphed. This column also serves a second purpose when errors are found

Setting	Description
	in the data. An exclamation point "!" character is placed in the column to signal an error, and direct you to look in the Messages column for the cause of the error.
`Item` Column	The column within the 'data' worksheet where the `Item` values are located.
`Label` Column	The column within the 'data' worksheet where the `Label` values are located.
`External Label` Column	The column within the 'data' worksheet where the `External Label` values are located.
`Tail Label` Column	The column within the 'data' worksheet where the `Tail Label` values are located.
`Head Label` Column	The column within the 'data' worksheet where the `Head Label` values are located.
`Related Item` Column	The within the 'data' worksheet column where the `Related Item` values will be found
`Style Name` Column	The Column within the 'data' worksheet where the `Style Name` name values are located.
`Attributes` Column	The column within the 'data' worksheet where the `Attributes` style attributes are located.
`Messages` Column	The column located within the 'data' worksheet where error messages are to be written to.
Heading Row	The row containing the column headings
First Row	First row of data.
Last Row	Last row of data definitions. When set to 0 all rows after the First Row containing data are used. First Row and Last Row are useful during graph development for specifying a subset of data to process when the overall amount of data is large.

'source' Worksheet

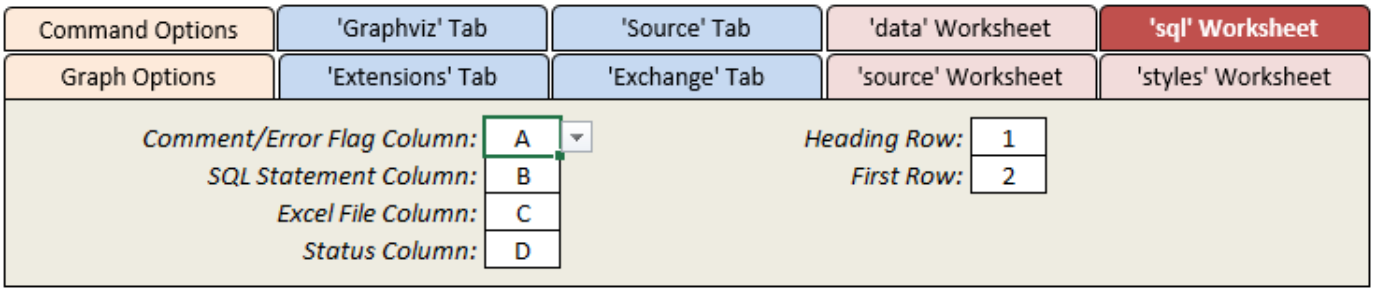
These settings allow you to rearrange the column layout of the source worksheet.



Setting	Description
`Line Number` Column	The column which contains the line number of the Graphviz DOT source code.
`Graphviz Source` Column	The column where the Graphviz DOT source code gets placed
Heading Row	The row containing the column headings
First Row	First row of data.
Tab Indent Spaces	The number of blanks to insert based upon the depth of subgraphs within the graph.

'sql' Worksheet

These settings allow you to rearrange the column layout of the sql worksheet, and restrict which rows get processed.

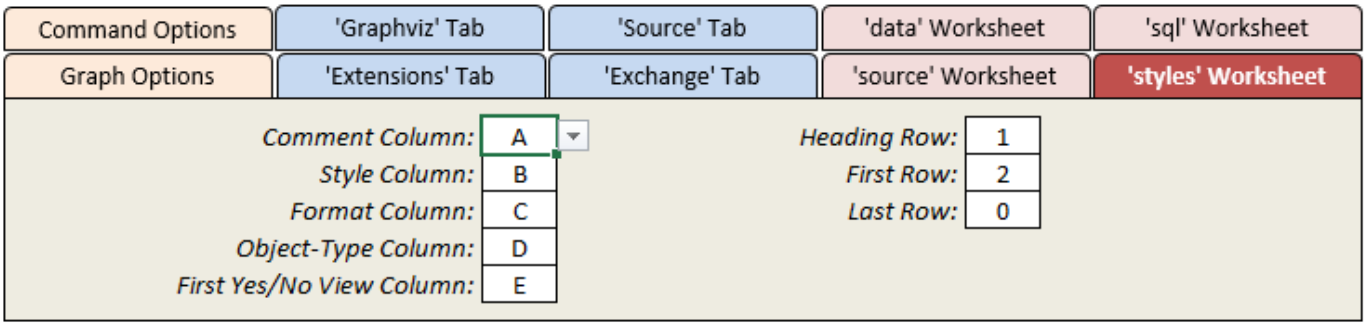


Setting	Description
`Comment/Error Flag` Column	The column used to place '#' comment characters which indicate that the rows should be treated as a comment and not be included in the set of SQL

Setting	Description
	statements which get executed.
`SQL Statement` Column	The column containing the SQL SELECT statements to execute
`Excel File` Column	The column containing the paths to the Excel files containing the data which the SQL SELECT statements should be executed against.
`Status` Column	The column where the status (Success/Failure/Skipped) of the SQL SELECT statement, along with any error messages is written to.
Heading Row	The row containing the column headings
First Row	First row of SQL SELECT statements to execute.

'styles' Worksheet

The 'styles' Worksheet' settings allow you to rearrange the column layout of the styles worksheet, and specify the range of rows to be included in the style collection at run-time.



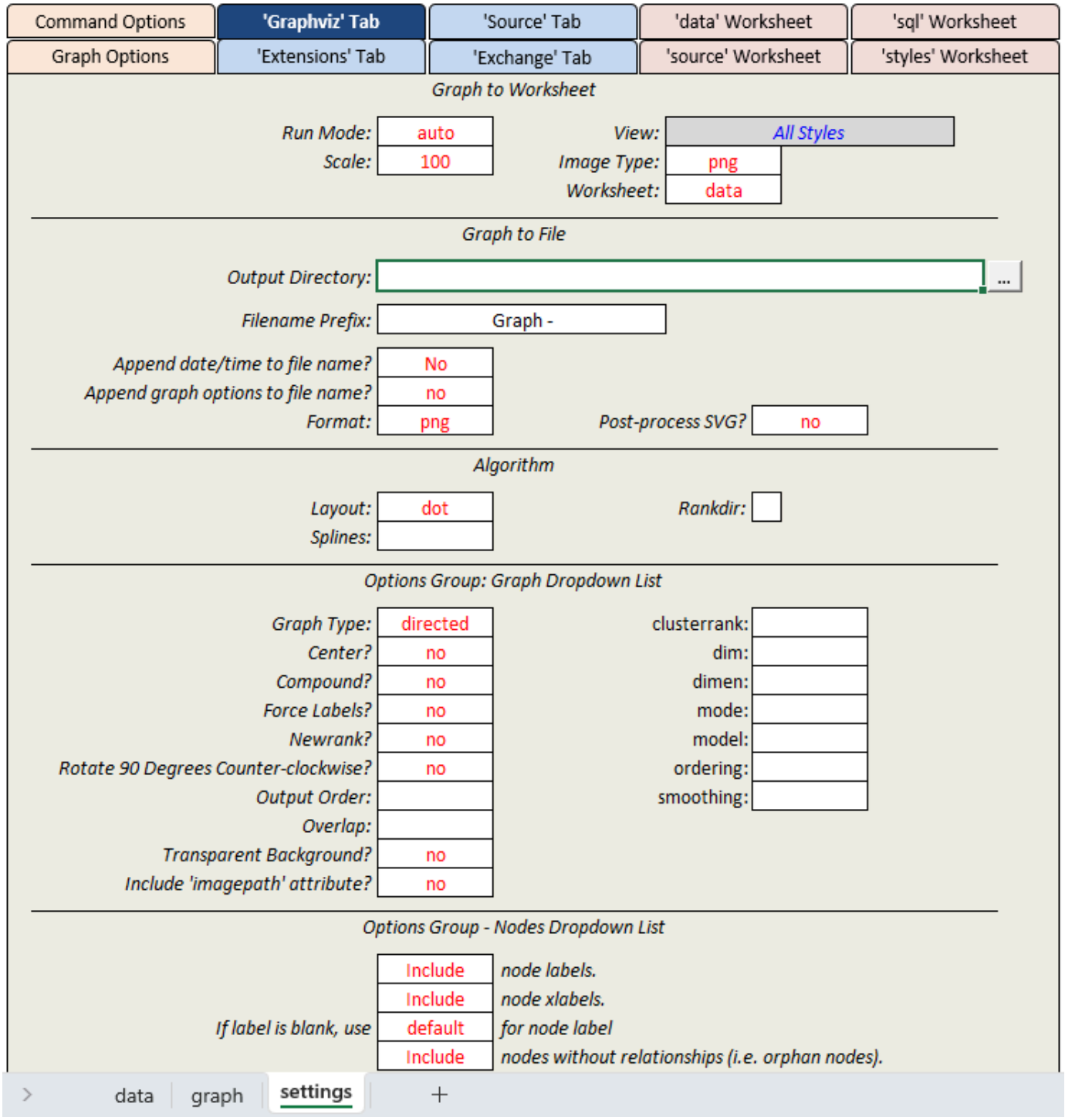
Setting	Description
`Comment` Column	The column used to place '#' comment characters which indicate that the rows should be treated as a comment and not be included in the set of styles.
`Style` Column	The column where the name of the style is specified
`Format` Column	The column where the node, edge, or cluster format string is specified.

Setting	Description
`Object-Type` Column	The column where the object-type is specified. The object-type is used to classify the style as a node, edge, subgraph-open, subgraph-close, keyword, or native type.
`First Yes/No View` Column	The first column containing Yes/No switches which control which styles will comprise the view.
Heading Row	The row which contains the column headings. The heading value of the "Yes/No Switch Column" is included in the graph's file name after the file name prefix.
First Row	First row of style definitions (i.e., where the definitions begin)
Last Row	Last row of style definitions. When set to 0 all rows after the `First Row` containing data are used.

Ribbon Tab Settings

'Graphviz' Tab

This worksheet stores the selected values from the Graphviz ribbon tab. By saving these settings in a worksheet, the Relationship Visualizer can retain your choices across sessions, ensuring that your preferred layout, styling, and rendering options persist each time you reopen the workbook.

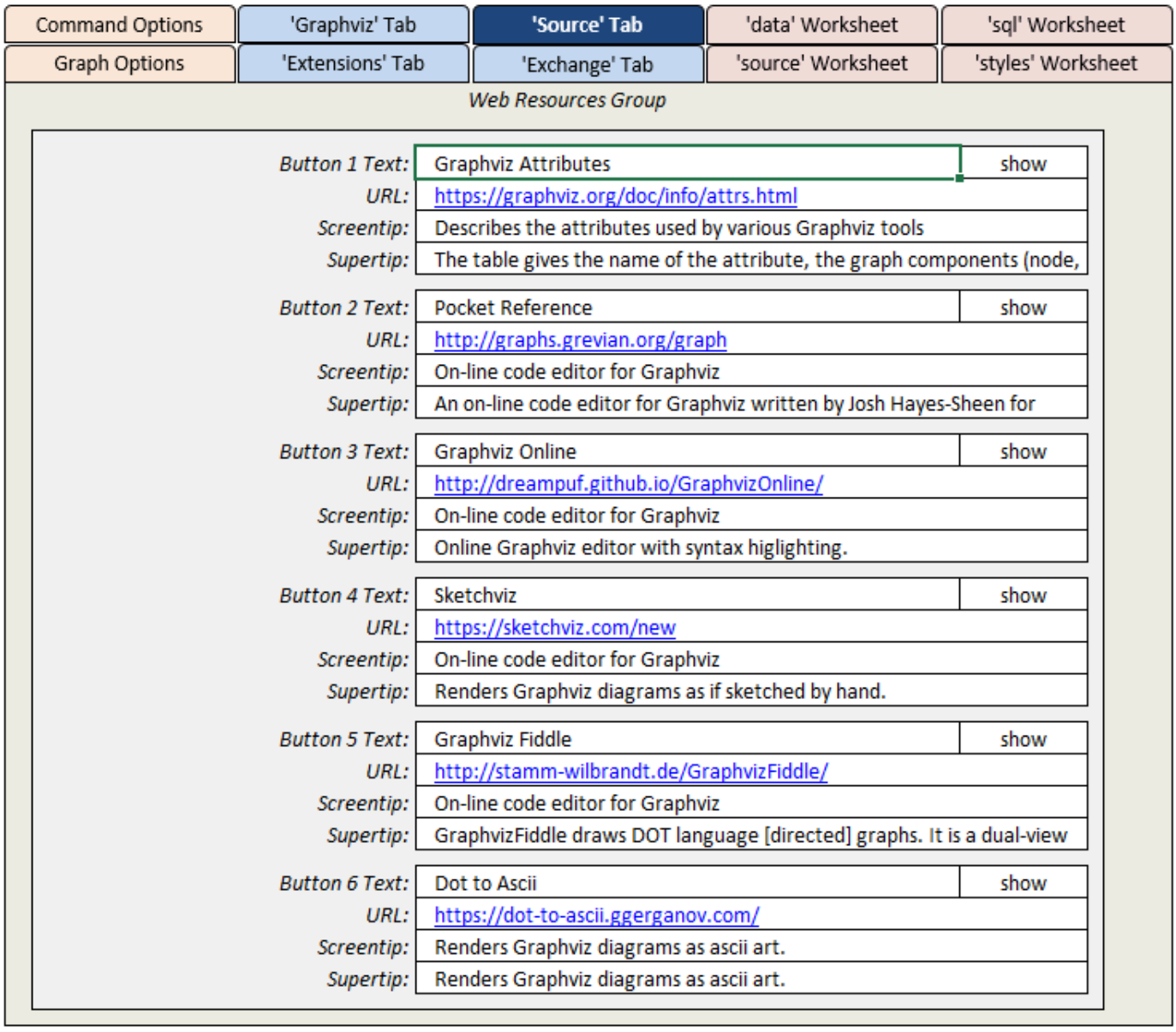


'Source' Tab

This worksheet stores the six web resources that appear as clickable links on the Source ribbon tab. The default URLs are suggestions provided by the Relationship Visualizer's author, but you are free to replace them with your own preferred resources. Updating the

links here ensures that the corresponding buttons on the `Source` tab always open the destinations you choose.

Typical substitutions include links to DOT language references, Graphviz attribute documentation, example galleries, internal documentation pages, or any external resources you find helpful while reviewing or editing the generated source code.



'Extensions' Tab

The `Extensions` worksheet is a seldom-used but powerful feature designed for expert users. It provides a way to add custom VBA procedures to the workbook and launch them

from the Ribbon without needing to understand the details of Ribbon XML or callback authoring. The feature supports up to six custom action buttons and six custom URL links.

A sample workbook demonstrating how to configure and use Extensions is included in the directory of examples.

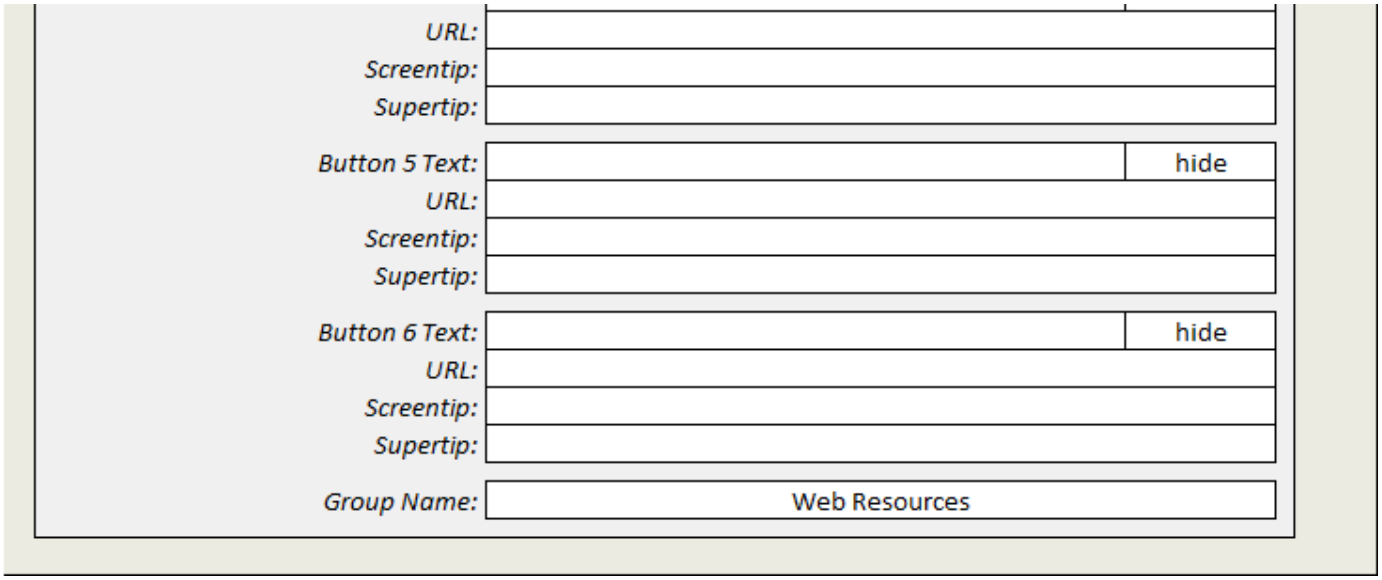
Tab Name: Extensions

Custom Code Group

Button 1 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Button 2 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Button 3 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Button 4 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Button 5 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Button 6 Text:		hide
Subroutine:		
Screentip:		
Supertip:		
Group Name:	Custom Code	

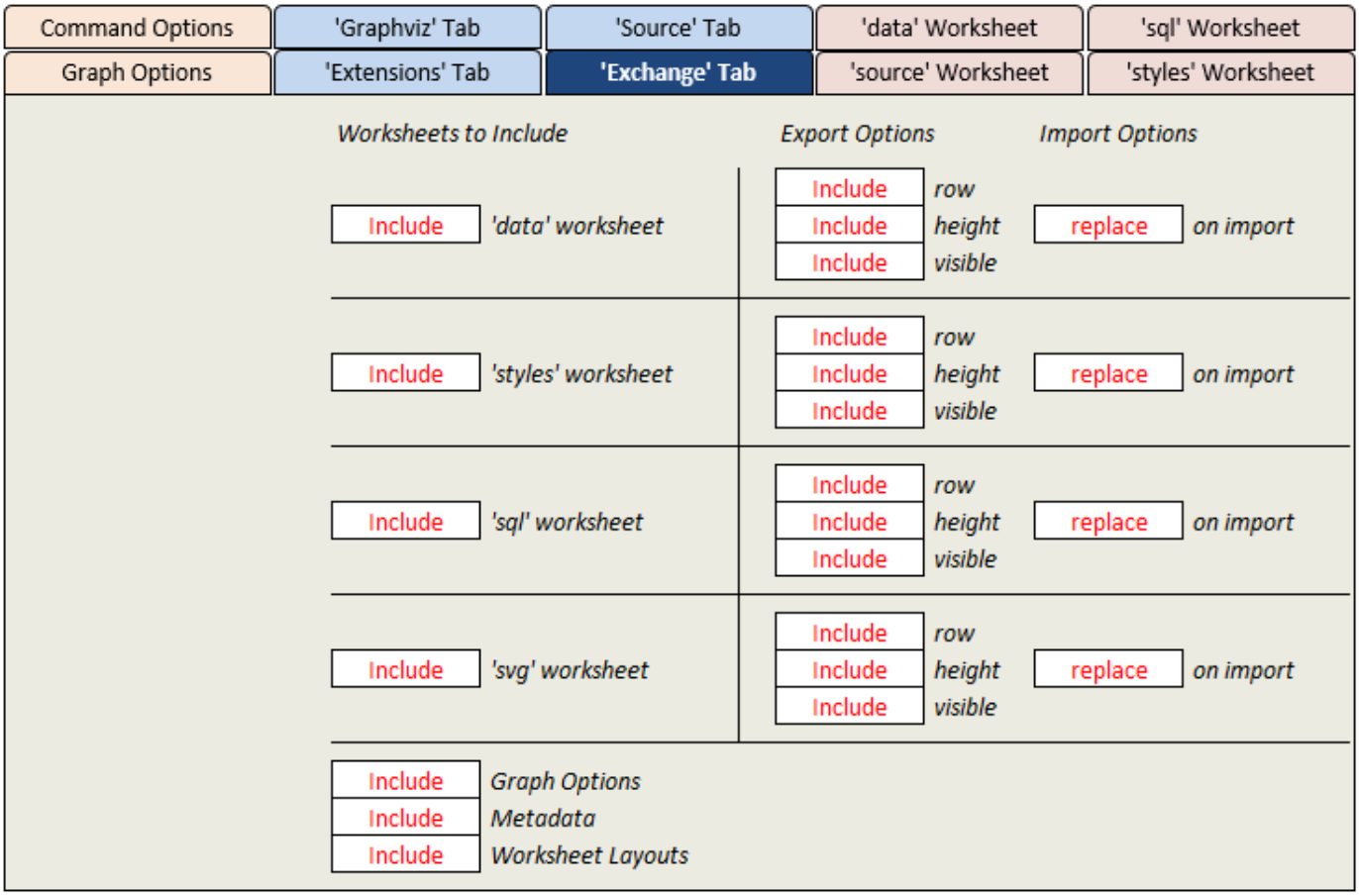
Web Resources Group

Button 1 Text:		hide
URL:		
Screentip:		
Supertip:		
Button 2 Text:		hide
URL:		
Screentip:		
Supertip:		
Button 3 Text:		hide
URL:		
Screentip:		
Supertip:		
Button 4 Text:		hide



'Exchange' Tab

This worksheet stores the selected values from the Exchange ribbon tab. By saving these settings in a worksheet, the Relationship Visualizer can retain your choices across sessions, ensuring that your preferred export, import, and workbook options persist each time you reopen the workbook.



Help Button URLs

The `Help Button URLs` worksheet defines the links used by the **Help** buttons on the various Ribbon tabs. Each row maps a specific Ribbon control to the URL that should open when the user selects its Help button.

This worksheet allows you to manage all Help destinations in one place. Updating a URL here ensures that the corresponding Help button on the Ribbon will always direct users to the correct documentation page, tutorial, or reference material.

Help Button URLs

'Graphviz' Tab:	https://exceltographviz.com/create/
'Style Designer' Tab:	https://exceltographviz.com/designer/
'Styles' Tab:	https://exceltographviz.com/styles/
'SQL' Tab:	https://exceltographviz.com/sql/
'SVG' Tab:	https://exceltographviz.com/svg/
'Console' Tab:	https://exceltographviz.com/console/
'Launchpad' Tab:	https://exceltographviz.com/launchpad/
'Source' Tab:	https://exceltographviz.com/source/
'Exchange' Tab:	https://exceltographviz.com/exchange/
'Diagnostics' Tab:	https://exceltographviz.com/diagnostics/
'Info' Tab:	https://exceltographviz.com/info/

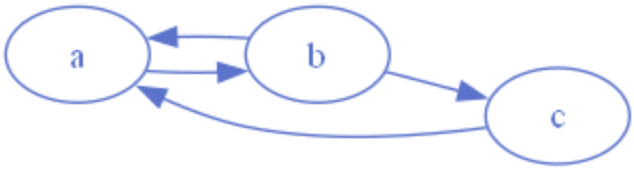
Terminology

In mathematics and computer science, **graph theory** is the study of graphs—mathematical structures used to model pairwise relationships between objects.

The terminology introduced here is used throughout the remainder of this website to describe how visualizations are built. These concepts originate in graph theory and the Graphviz tool.

Graph

A **graph** is a collection of nodes and the edges that connect them. The illustration below shows a simple example of a graph.



Node

A graph is composed of **nodes**, which represent the objects or entities in the visualization.

Edge

Edges are the connections between nodes. They represent the relationships or interactions between the entities.

Undirected Graph

An **undirected graph** is one in which edges have no inherent direction.
The relationship between the two connected nodes is mutual.

Directed Graph

A **directed graph** (or **digraph**) uses edges with explicit direction, indicating a one-way relationship from one node to another.

Labels

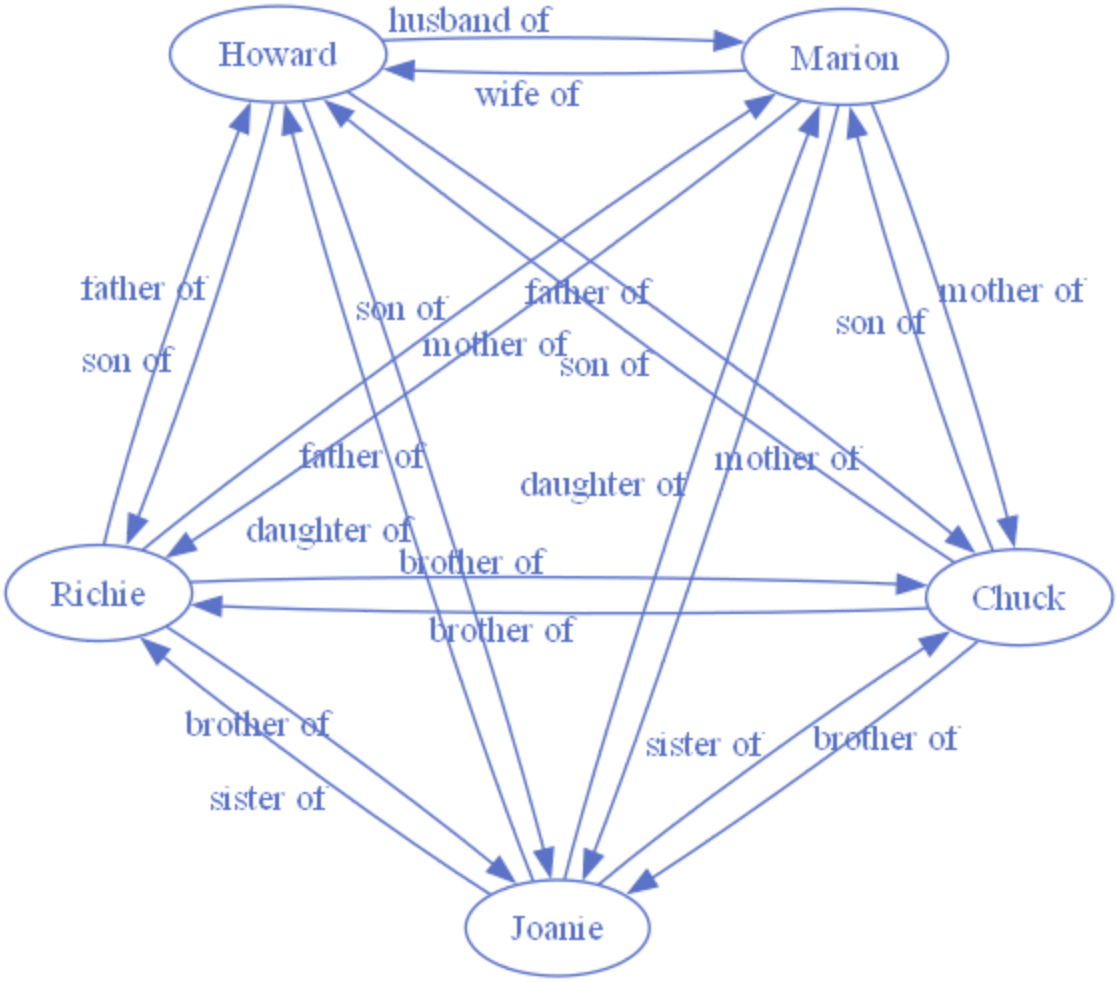
Nodes can have **labels**, which provide text describing the node. Labels may appear **inside** the node or **outside** the node, depending on the node shape and the label attributes.

Edges can also have labels. The main edge label appears **on the edge** itself:

Labels may also be placed at the **tail** and/or **head** of the edge using the `taillabel` and `headlabel` attributes:

A label can also be positioned **outside** the edge using the `xlabel` attribute, although this placement may vary depending on the layout algorithm and spline routing:

Edge labels are especially useful for describing the **relationship** between nodes. For example, a set of family relationships might be labeled as follows:



Splines

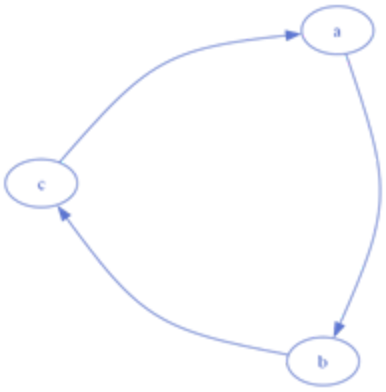
The way edges are routed and drawn in a graph is controlled by the **splines** setting. Graphviz supports several spline types, each producing a different style of edge routing.

The available options are described below.

curved

Edges are drawn as smooth, continuous curves between nodes.

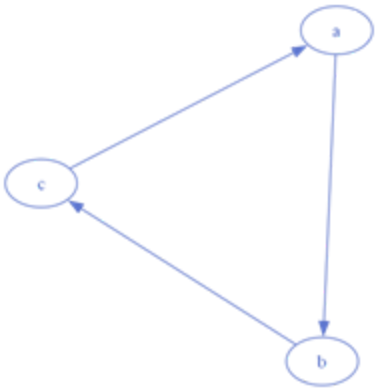
This style produces flowing, organic connections.



line

Edges are drawn as single straight lines between nodes.

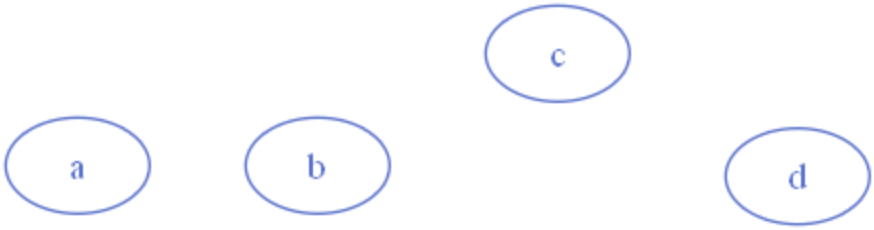
This is the simplest and most geometric routing style.



none

Edges (and edge labels) are **not drawn**, but the underlying relationships still influence node placement.

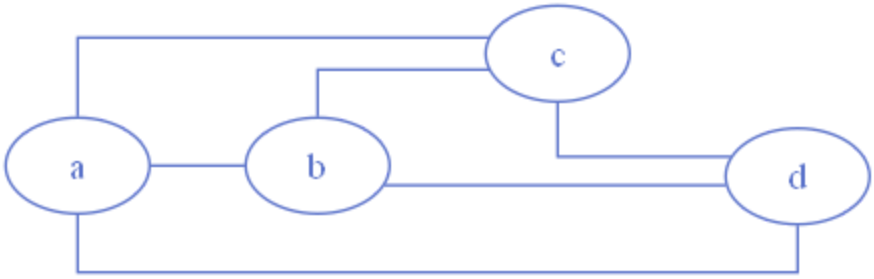
Useful when you want layout guidance without visible edges.



ortho

Edges are routed using horizontal and vertical segments with **90-degree bends**.

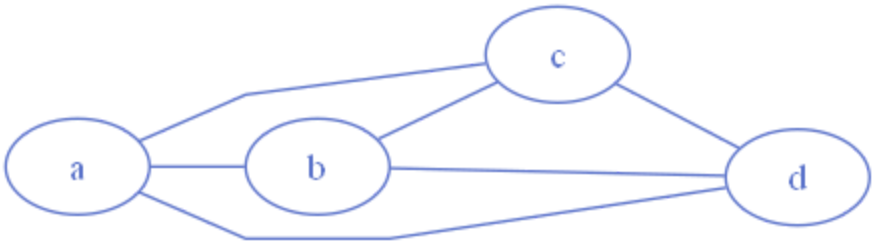
Ideal for diagrams that benefit from clean, rectilinear structure.



polyline

Edges are drawn as straight segments with **angular bends**.

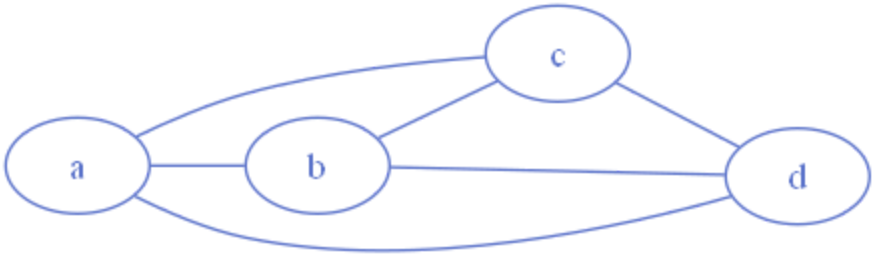
This style preserves sharp turns without enforcing strict right angles.



spline

Edges are drawn using a combination of straight segments and **free-flowing curves**.

This is the most flexible routing style and often reduces visual clutter in dense graphs.



Ports

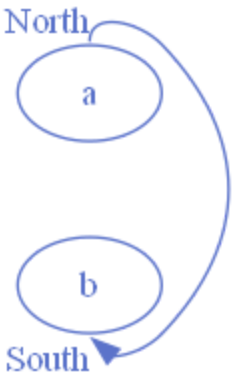
A **port** can be combined with a node name to specify exactly where an edge should attach to that node.

Graphviz provides several built-in port names that correspond to compass points:

Port Name	Represents
`c`	Center
`n`	North
`s`	South
`e`	East
`w`	West

Port Name	Represents
`ne`	North-East
`nw`	North-West
`se`	South-East
`sw`	South-West

Ports allow you to control the entry and exit points of edges, which can help reduce crossings or emphasize directional flow.



Custom ports can also be defined when using HTML-like labels or the record node shape.

This allows edges to connect to specific fields or sub-components within a node.

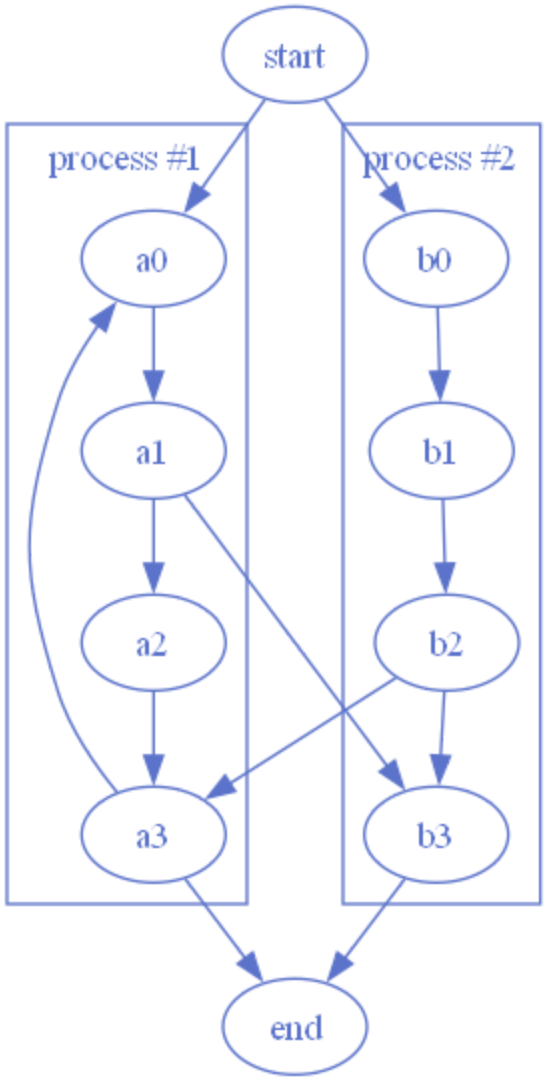
Clusters / Subgraphs

A **cluster** is a special type of subgraph that groups related nodes and edges inside a distinct rectangular region.

Clusters are defined as subgraphs within the parent graph, and Graphviz automatically draws a bounding box around each one.

Only the dot, fdp, neato, and osage layout engines support visual cluster rendering.

In the example below, the rectangles labeled **process #1** and **process #2** are clusters (subgraphs) contained within the overall graph.



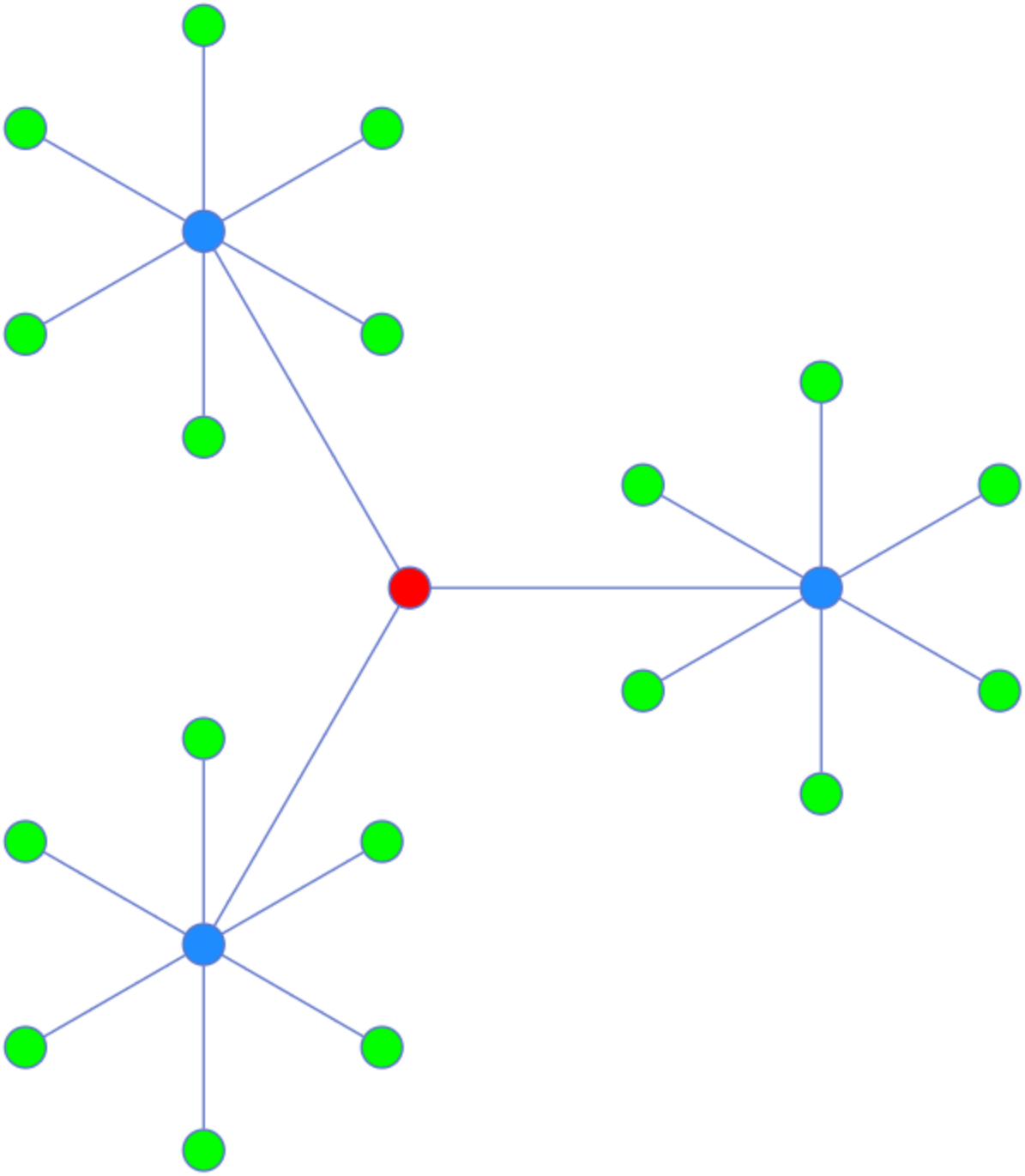
Layout Algorithms

Graphviz provides several layout engines, each using a different algorithm to position nodes and route edges. Some layouts are optimized for hierarchical structures, others for clusters, radial patterns, or force-directed placement. Choosing the best layout is often a matter of experimentation to see which representation communicates your data most effectively.

Descriptions of the available layout engines (based on the Graphviz documentation) are as follows:

circo

Circular / Cyclic



Produces **circular** layouts.

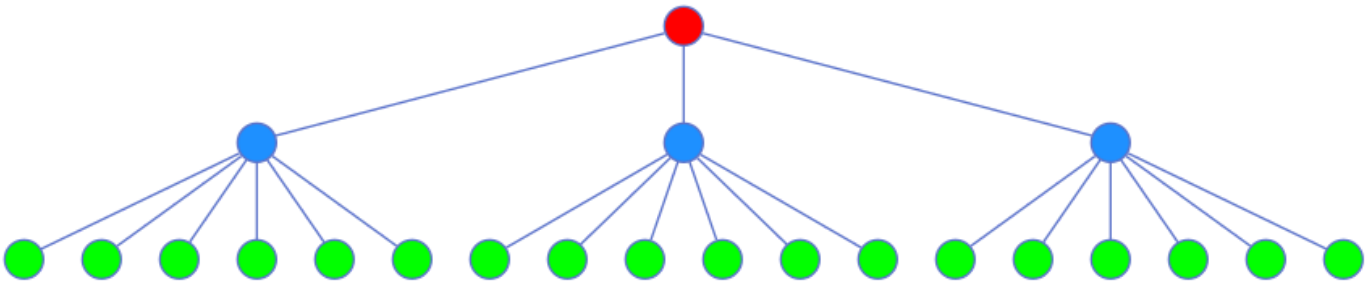
Designed for graphs with multiple cyclic structures and is useful for telecommunications diagrams or any network with repeating loops.

- Best for: Cycles, rings, telecommunications networks
- Strengths: Emphasizes circular symmetry
- Avoid for: Hierarchies or tree structures

---

dot

Hierarchical / Layered



Creates **hierarchical** or **layered** drawings of directed graphs.

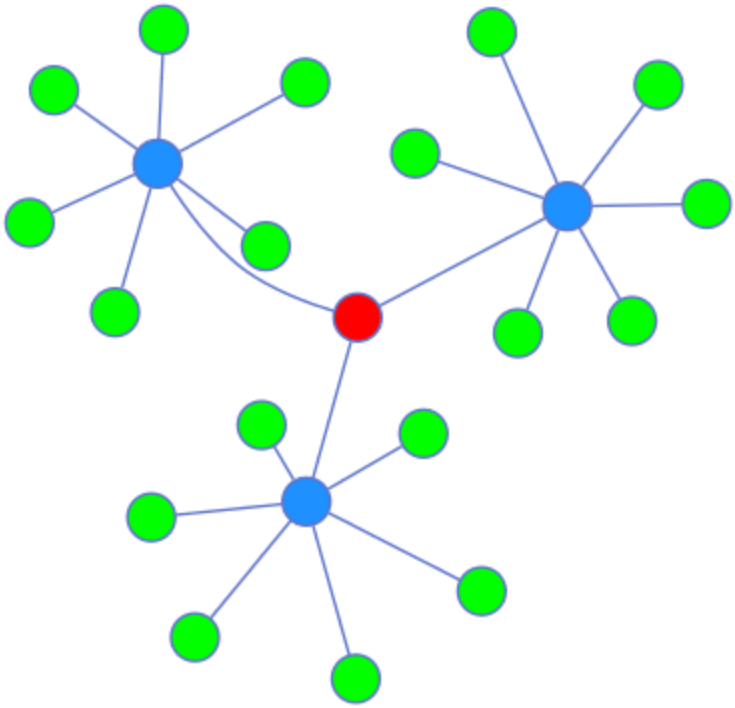
Ideal when you want clear top-to-bottom or left-to-right flow, making it the default choice for most structured diagrams.

- Best for: Directed graphs, workflows, org charts, dependency trees
- Strengths: Clear direction (TB, LR), predictable structure
- Avoid for: Dense undirected networks

---

fdp

Spring Model (Force Reduction)



A **force-directed placement** algorithm similar to *neato*, but it reduces forces directly rather than minimizing an energy function.

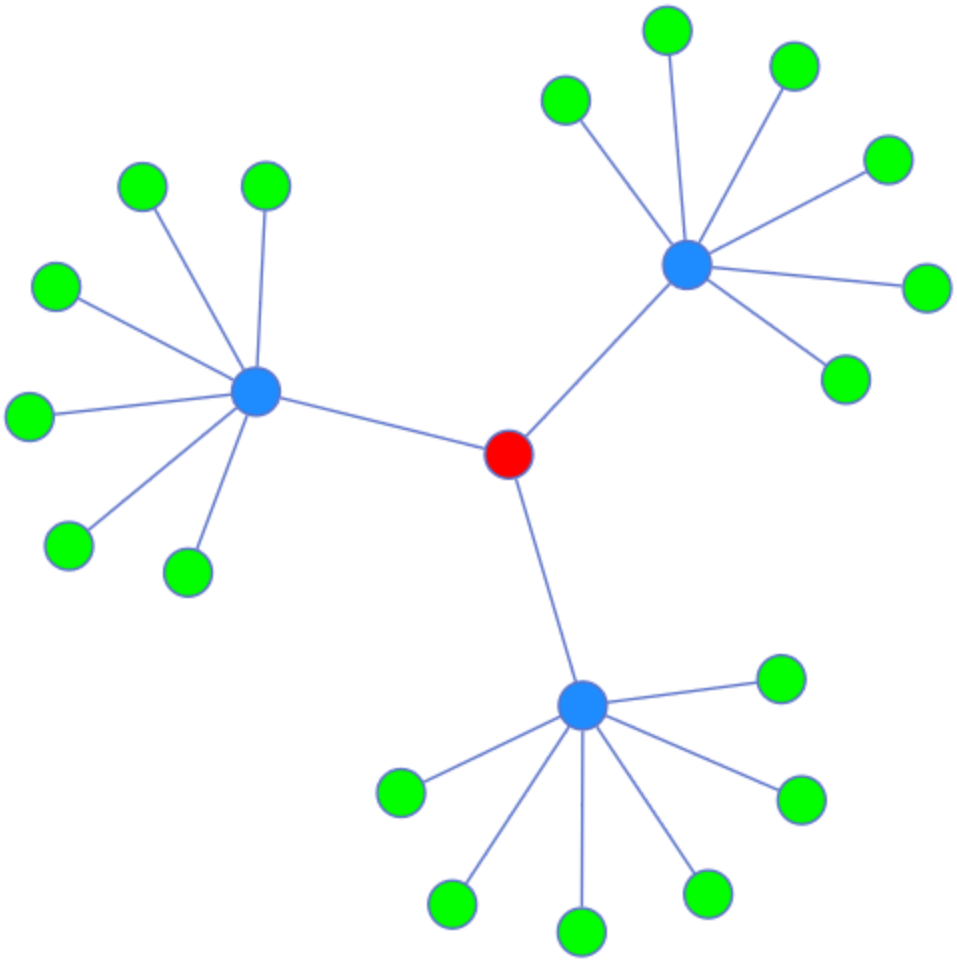
Useful for undirected graphs where a natural, organic layout is desired.

- Best for: Undirected graphs needing organic spacing
- Strengths: Similar to neato but uses force reduction
- Avoid for: Very large graphs (use sfdp instead)

---

neato

Spring Model (Energy Minimization)



A classic **spring-model** layout engine.

Best for small to medium-sized undirected graphs (roughly up to 100 nodes).

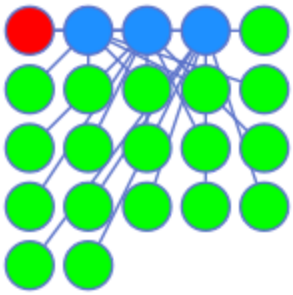
Neato attempts to minimize a global energy function, producing layouts similar to multidimensional scaling.

- Best for: Small–medium undirected graphs (~100 nodes)
- Strengths: Balanced, natural layouts
- Avoid for: Very large graphs

---

osage

Clustered Rectangular Packing



Designed for **clustered** or **multi-level** undirected graphs.

Osage arranges clusters into rectangular regions (“levels”) and then packs these regions together.

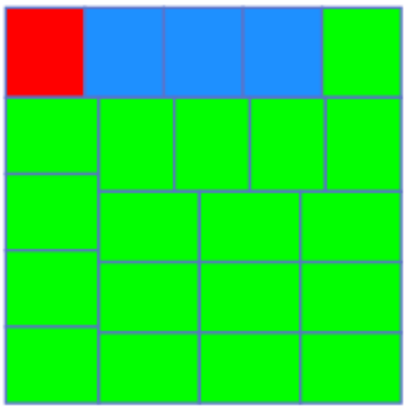
Within each region, the subgraph is laid out independently.

- Best for: Multi-level clustered graphs
- Strengths: Separates clusters into rectangles and packs them
- Avoid for: Flat, non-clustered graphs

---

patchwork

Squarified Treemap



Draws the graph as a **squarified treemap**, using cluster structure to determine the hierarchy.

Ideal for visualizing nested groupings or proportional areas.

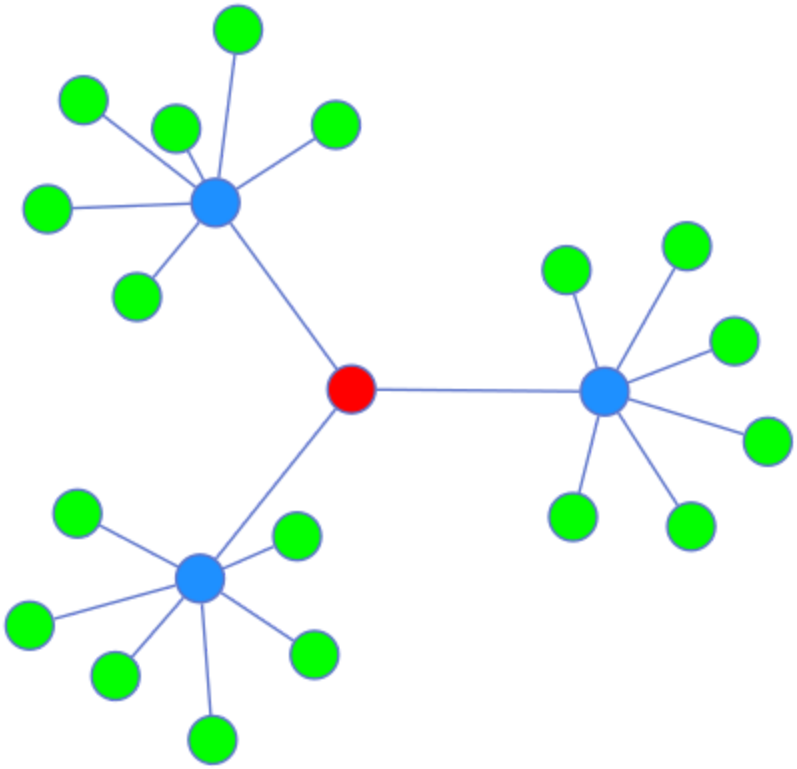
- Best for: Hierarchical clusters, proportional areas

- Strengths: Treemap-style visualization
- Avoid for: Non-clustered graphs

---

sfdp

Multiscale Force-Directed



A **multiscale** version of *fdp*, optimized for **very large** graphs.

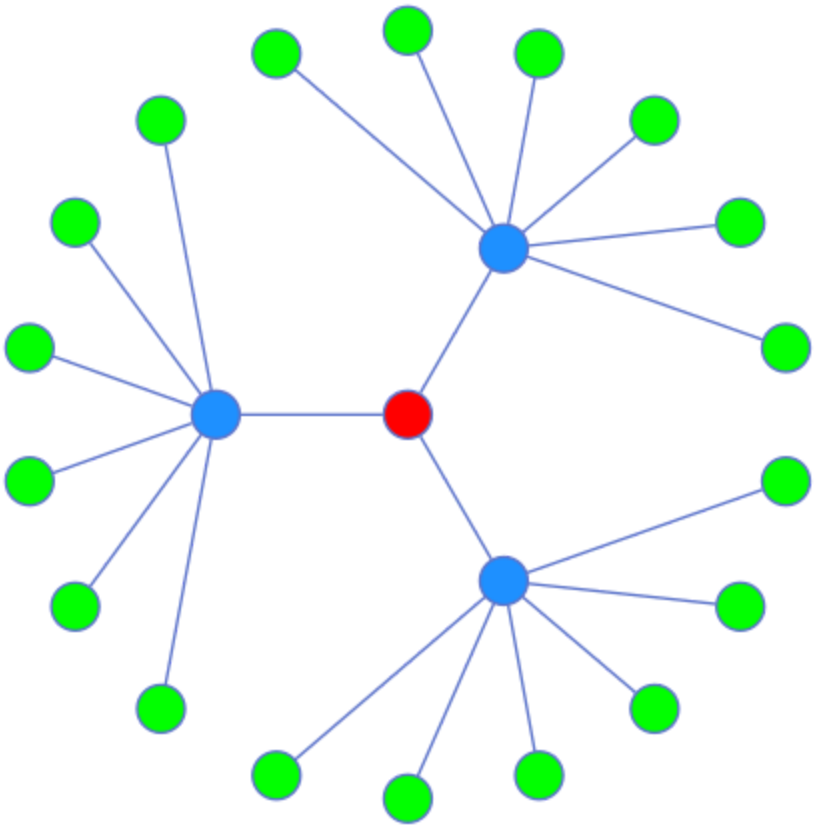
It uses a hierarchical approximation to compute force-directed layouts efficiently at scale.

- Best for: Very large undirected graphs
- Strengths: Scales efficiently, good for thousands of nodes
- Avoid for: Small graphs (overkill)

---

twopi

Radial / Concentric Circles



Produces **radial** layouts.

Nodes are placed on concentric circles based on their distance from a chosen root node, making it ideal for tree-like structures or distance-based visualizations.

- Best for: Rooted trees, distance-based layouts
- Strengths: Clear radial structure
- Avoid for: Dense or cyclic graphs

Layout Engine Comparison

Layout Engine	Primary Style / Algorithm	Best For	Not Ideal For	Notes
circo	Circular / cyclic placement	Cycles, rings, telecommunications networks	Strict hierarchies or	Emphasizes repeated loops

Layout Engine	Primary Style / Algorithm	Best For	Not Ideal For	Notes
			tree structures	and circular symmetry
dot	Hierarchical / layered	Directed graphs, workflows, dependency trees	Dense undirected networks	Most control over direction (TB, LR, etc.)
fdp	Force-directed (force reduction)	Undirected graphs needing organic spacing	Very large graphs (use sfdp instead)	Similar to neato but uses force reduction rather than energy minimization
neato	Force-directed (energy minimization)	Small–medium undirected graphs (~100 nodes)	Large graphs or strict hierarchies	Produces layouts similar to multidimensional scaling
osage	Cluster-level rectangular packing	Multi-level clustered graphs	Non-clustered or flat graphs	Lays out each cluster in its own rectangle, then packs them
patchwork	Squarified treemap	Hierarchical clusters, proportional areas	Non-clustered graphs	Uses cluster structure to build a treemap-style layout
sfdp	Multiscale force-directed	Very large undirected graphs	Small graphs (overkill)	Scales well using hierarchical approximations
twopi	Radial / concentric circles	Rooted trees, distance-based layouts	Dense or cyclic graphs	Places nodes on circles based on distance from a root

Quick-Reference: Choosing a Graphviz Layout

Use this guide to quickly select the layout engine that best fits your graph's structure and purpose.

If your graph is...

- **Hierarchical or directional** → Use **dot**
  Clear top-to-bottom or left-to-right flow; ideal for workflows, org charts, dependency trees.
- **Small–medium and undirected** → Use **neato**
  Produces balanced, natural layouts using a spring-model energy function.
- **Undirected and organic-looking** → Use **fdp**
  Similar to neato but uses force reduction; good for medium-sized networks.
- **Very large and undirected** → Use **sfdp**
  Multiscale force-directed algorithm optimized for thousands of nodes.
- **Cyclic or ring-structured** → Use **circo**
  Best for loops, cycles, and circular network patterns.
- **Clustered into multiple groups** → Use **osage**
  Places each cluster in its own rectangle and packs them together.
- **Hierarchical clusters or treemap-style visuals** → Use **patchwork**
  Converts cluster structure into a squarified treemap layout.
- **Rooted or distance-based** → Use **twopi**
  Radial layout placing nodes on concentric circles around a root.

Quick Tips

- **Start with dot** for anything directional.

- **Try neato or fdp** for undirected graphs when you want a natural look.
- **Switch to sfdp** if the graph becomes too large or slow.
- **Use circo** when cycles are the main feature.
- **Use osage or patchwork** when clusters matter more than edges.
- **Use twopi** when distance from a root node is the key story.

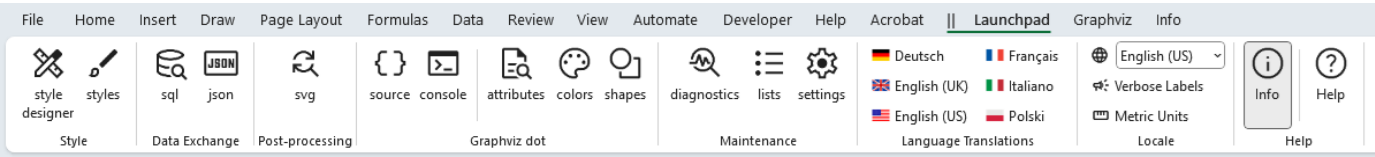
When in doubt

Generate previews with **dot**, **neato**, and **sfdp** first — these three cover most practical cases and quickly reveal which style fits your data.

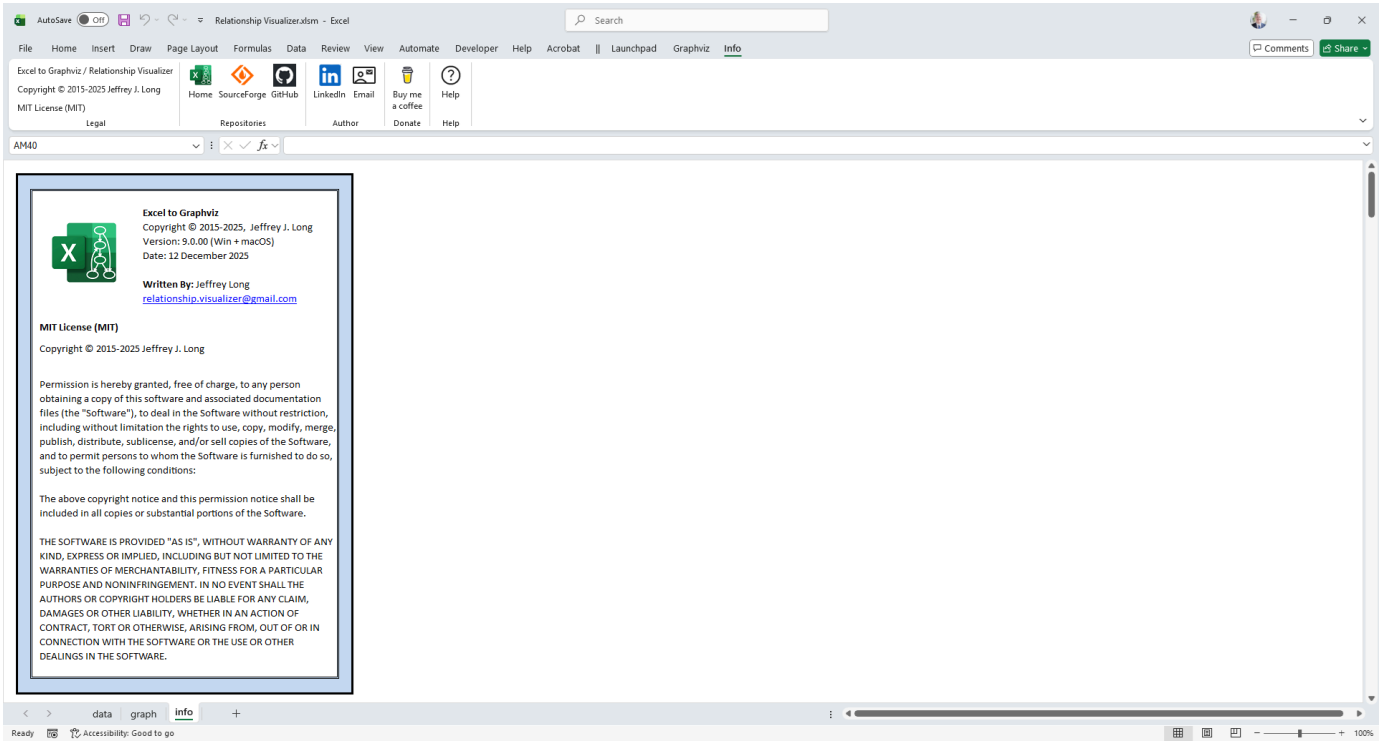
Info Worksheet

The `info` Worksheet

The `info` worksheet is reached from the `Help` section of the Launchpad ribbon tab.



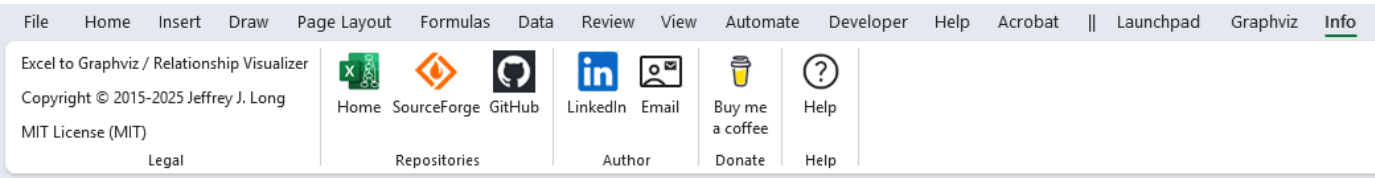
The `info` worksheet provides information about the spreadsheet, the internal technology incorporated into the tool, and the associated Open Source licenses and acknowledgements required to reuse the Open Source components in a license-compliant manner.



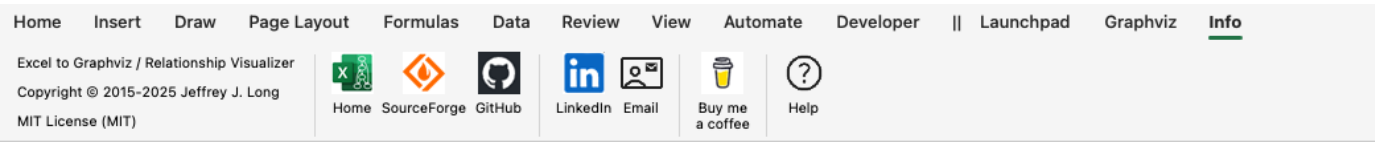
The `Info` Ribbon Tab

The `Info` ribbon tab is activated whenever the `Info` worksheet is activated. It appears as follows:

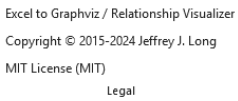
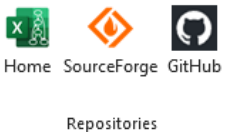
`Windows`



`macOS`

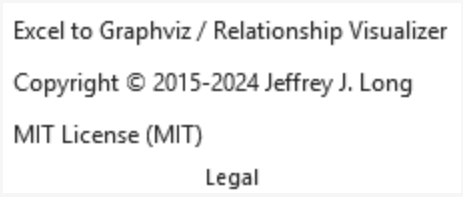


It contains the following groups, each of which is explained in the sections that follow. You may jump directly to any group using the links in this table:

Group	Controls	Description
Legal		The `Legal` section provides a quick overall summary of the tool's name, copyright, and Open Source license.
Repositories		The `Repositories` section provides web links to the **official** web pages and repositories for the Relationship Visualizer tool.
Author		Contact information for reaching the author of this tool.

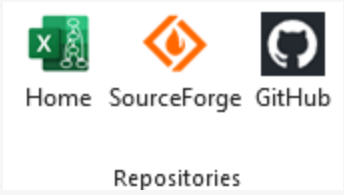
Group	Controls	Description
Donate	 Buy me a coffee Donate	If you would like to show your appreciation for this **free** tool, you can buy the author a coffee through the Buy Me A Coffee ↗ website.
Help	 Help Help	Provides a link to the `Help` content for the `Info` worksheet (i.e. this web page).

Legal



The `Legal` section provides a quick overall summary of the tool's name, copyright, and Open Source license.

Repositories



The `Repositories` section provides web links to the **official** web pages and repositories for the Relationship Visualizer tool.

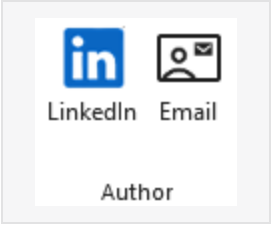
Download Only from Official Sources

At this time, there are **no** author-sanctioned clones of these repositories. However, unofficial copies have been found on the internet. If you obtained your copy of the Relationship Visualizer

from anywhere other than the sites linked by these buttons, you may be using an out-of-date or potentially security-compromised version.

Label	Control Type	Description
Home	Button	Provides a link to the Excel To Graphviz ↗ documentation web site (i.e. this web site).
SourceForge	Button	Provides a link to the SourceForge ↗ repository where the Relationship Visualizer download files are managed.
GitHub	Button	Provides a link to the GitHub ↗ repository where the source pages for this website are maintained. The VBA source code from the Relationship Visualizer workbook is also exported and stored there. Although the workbook cannot be rebuilt from this source code, publishing it in a readable, text-based format ensures full transparency and allows for inspection.

Author



If you would like to learn more about the author of this tool and website you may visit his LinkedIn profile. You can also reach him by email.

Label	Control Type	Description
LinkedIn	Button	Provides a link to the author's LinkedIn profile.

Label	Control Type	Description
Email	Button	Launches your email client and creates a draft email which you can send to the author.

Donate

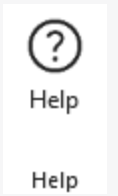
Excel to Graphviz / Relationship Visualizer is **free** software. However, it does cost the author personal money and time to provide this software for free.

If you would like to show your appreciation for this **free** tool, you can buy the author a coffee through the Buy Me A Coffee ↗ website.

FYI - 10 years, 10,000+ downloads, and a grand total of 2 coffees donated 😔

Label	Control Type	Description
Donate	Button	Provides a link to the Buy Me A Coffee ↗ website where you can show your appreciation to the author for this free software and awesome web site.

Help



Provides a link to the `Help` content for the `Info` worksheet (i.e. this web page).

Label	Control Type	Description
Help	Button	Provides a link to this web page.

Security

Excel Macro-Enabled Workbook Considerations

When deciding whether to trust a macro-enabled Excel workbook (such as an `.xlsm` file) downloaded from the Internet, several security factors should be evaluated to minimize risk. Macro-enabled workbooks contain executable VBA code, which can pose significant security threats if malicious or poorly vetted.

The sections below outline the key considerations that help you make an informed, cautious decision before enabling macros or running downloaded Excel tools, and how these considerations apply specifically to the `Relationship Visualizer.xlsm` spreadsheet.

Assess Reputation and Trustworthiness

Verify the **reputation of the repository owner**. Review their profile for activity history, contributions, and community engagement. Well-established or transparent contributors are generally more trustworthy.

- A public professional profile for the author and repository owner is available on LinkedIn ↗.

Review Repository Activity

Review the project's Changelog, which documents the history of its releases and provides insight into development cadence and maintenance quality.

Evaluate the repository's visibility and engagement. A well-maintained project with stars, forks, active issues, and community participation is less likely to host malicious or

abandoned code.

- The *Excel to Graphviz Relationship Visualizer* has published releases on **SourceForge** continuously since **October 17, 2015**.
- It holds a **five-star rating** on SourceForge.
- SourceForge has awarded it a **Community Choice** badge, recognizing open-source projects that have reached the milestone of **10,000 total downloads**.

Check for **code reviews**, comments, or discussions in the repository.

- Contributions from multiple users or public conversations can indicate scrutiny, transparency, and reliability.

Review the project's Issues Log ↗.

- **Open issues** may highlight concerns raised by the community or known software defects.
- **Closed issues** provide insight into past problems and whether they were resolved promptly and responsibly.

Inspect the Code

Review the macro code before enabling it.

- The VBA macro source code for the *Excel to Graphviz Relationship Visualizer* workbook is published on GitHub at

  https://github.com/jjlong150/ExcelToGraphviz/tree/main/src ↗

  It can be viewed directly in a web browser before downloading or opening the workbook.

Open the workbook in a protected environment and inspect the VBA code (Alt+F11 in Excel). Look for suspicious or unexpected actions such as:

- **Network connections** (e.g., accessing external URLs).

  - The *Excel to Graphviz Relationship Visualizer* includes hyperlinks to online help resources and Graphviz rendering sites, but does not perform automated network calls.

- **File system modifications** (e.g., creating, deleting, or altering files).
  - The workbook generates text and image files as part of its normal operation.
- **Execution of external programs or scripts.**
  - The workbook invokes the Graphviz `dot` command-line program to convert `.dot` source files into rendered graphs.

If you lack VBA expertise, **consider using online tools or sandbox environments** to analyze the code for malicious behavior.

All macros in the *Excel to Graphviz Relationship Visualizer* undergo static code analysis using RubberduckVBA ↗ prior to publication. Rubberduck provides inspections, code metrics, and quality checks that help ensure maintainability and reduce the likelihood of hidden or unsafe behavior.

Verify Files

One of the most important security steps is knowing **exactly where the file came from**. Avoid downloads from unauthorized mirrors, repackaged distributions, or any file not obtained directly from the official project pages, as these sources may compromise integrity or security.

Use only official releases

Official releases of the *Excel to Graphviz Relationship Visualizer* workbook are published exclusively on:

- **SourceForge** → https://sourceforge.net/projects/relationship-visualizer/ ↗
- **GitHub** → https://github.com/jjlong150/ExcelToGraphviz/releases ↗

Protect from viruses

- SourceForge performs automatic scanning before publishing files.
- GitHub does not scan release assets, so user-side scanning is essential.

Platform	Virus Scanning of Uploaded Files	Notes
SourceForge	✔ Yes — automatic antivirus scanning on upload	Files are only made available for download **after** passing a successful virus scan.
GitHub	❌ No — GitHub does *not* scan release assets	GitHub security tools (Dependabot, code scanning) apply to **source code**, not binary release files. Users must scan downloaded files themselves.

Regardless of platform, always scan downloaded files with **reputable antivirus tools**.

Check file hash values

Verifying published hash values ensures the file you downloaded is **identical** to the one the author released.

Platform	Hash Types Provided	Where to Find Them	Notes
SourceForge	**SHA1**, **MD5**	On the **Files** tab → click the circle **(i)** icon next to each file	Hashes are shown for every downloadable artifact.
GitHub	**SHA256**	In the **Assets** section of each release	Hashes accompany ZIP files containing the workbook and source code.

Why verify hashes?

Matching the published hash confirms the file has not been altered, corrupted, or tampered with. Even a one-byte change produces a completely different hash.

Unblock internet files with caution

When you download an executable or ZIP file on Windows, the system may mark it as coming from the internet and block it from running. This is a safety measure: **Windows treats downloaded files as potentially unsafe** until you explicitly confirm you trust them. Macro-enabled Excel workbooks fall into this category as well.

How to unblock a file

1. Right-click the downloaded file.
2. Select **Properties**.
3. In the **General** tab, look for an **Unblock** checkbox near the bottom (it appears only if the file is blocked).
4. Check **Unblock** to remove the restriction.
5. Click **OK** or **Apply**.

If you don't see the **Unblock** option, the file may already be unblocked, or other security layers (administrator permissions, antivirus, or SmartScreen) may be preventing it from running.

WARNING

Unblocking a ZIP file does **not** unblock the files inside it.
You may need to unblock the extracted workbook separately.

After extracting the ZIP, you may need to repeat the **Unblock** process for the extracted workbook or any executable files it contains.

Why Windows preserves the "Mark of the Web" inside ZIP archives

Windows attaches a *Mark of the Web* (MOTW) to downloaded files to indicate they originated from an untrusted zone (e.g., the Internet). When a ZIP file carries this mark:

- Windows propagates the MOTW to **every file extracted** from the ZIP.
- Each extracted file is treated as if it were downloaded individually from the Internet.
- Excel uses the MOTW to enforce **stricter macro security**, including:
  - Blocking macros by default,
  - Showing security warnings,
  - Requiring explicit user trust before enabling code.

This behavior is intentional: it prevents malicious ZIP archives from bypassing security simply by packaging harmful files inside a container. As a result, even after unblocking the ZIP itself, you may still need to unblock the extracted workbook before Excel will allow macros to run.

Excel Security Settings

Ensure Excel's macro settings are configured to **disable macros by default** with a prompt to enable them (Trust Center → Macro Settings). This prevents macros from running automatically.

Avoid enabling macros unless you fully trust the file and its source. If Excel prompts you to enable macros, proceed thoughtfully.

Excel also supports **Trusted Locations** which are designated folders where macro-enabled spreadsheets can open without prompting. This lets you work with your own tools and automation safely, *without* loosening security for all files. Only add folders you control (for example, a local development directory), and avoid marking broad system folders or shared network paths as trusted.

How to configure a Trusted Location

1. Open **File** → **Options** → **Trust Center**.
2. Click **Trust Center Settings...**.
3. Select **Trusted Locations**.
4. Click **Add new location...**.

5. Choose a folder you control (e.g., a dedicated “ExcelTools” directory).
6. Confirm and close the dialog.

Once added, any macro-enabled workbook stored in that folder will open without security prompts while all other files remain protected.

Test in an Isolated Environment

Open the workbook in a **sandboxed or isolated environment**, such as a virtual machine (VM) or a dedicated device not connected to sensitive networks or data. This limits potential damage if the macro is malicious.

Alternatively, use a **cloud-based or disposable environment** (e.g., Windows Sandbox) to test the file.

Review Purpose and Necessity

Evaluate whether the workbook’s functionality **requires macros**. If the workbook can be used without enabling macros, it’s safer to leave them disabled.

- The *Excel to Graphviz Relationship Visualizer* workbook is fully dependent on enabled macros.
- If you cannot enable macros, the software cannot operate.

Review Documentation

Check whether the repository provides **clear documentation** explaining the macro’s purpose and functionality. A lack of transparent, well-structured documentation is a red flag.

- Comprehensive end-user documentation is published at https://exceltographviz.com ↗.

- The *Excel to Graphviz Relationship Visualizer* source code is fully documented, though the codebase contains thousands of lines of VBA and the comments are primarily written for maintenance and support.
- In addition, the project now includes DeepWiki-generated documentation at https://deepwiki.com/jjlong150/ExcelToGraphviz ↗, offering structured, cross-linked explanations of modules, workflows, and internal architecture to help users and developers understand how the system operates.

Check Community Feedback

Look for **user feedback** in the project's issue trackers, discussions, and social channels. Reports of suspicious behavior, unexpected macro activity, or security concerns should always be taken seriously.

- **SourceForge** → Support Tickets ↗
- **GitHub** → Issues ↗
- **X / Twitter** → @ExcelToGraphviz ↗
- **External sources** → Forums, blogs, and community discussions such as the Graphviz Forum ↗ may also provide independent feedback or warnings.

Is it Being Maintained?

Confirm that the repository is **actively maintained**. Abandoned projects or outdated files may contain unpatched vulnerabilities or rely on deprecated dependencies.

- The *Excel to Graphviz Relationship Visualizer* has published:
  - Downloadable runtime files on SourceForge ↗ continuously since **October 17, 2015**.
  - Exported VBA code and website markdown content on GitHub ↗ since **February 11, 2022**.
- Look for recent commits, releases, or updates that address bugs, compatibility issues, or security concerns. Active issue responses and ongoing documentation updates

(including DeepWiki-generated content) are also strong indicators of healthy maintenance.

Assess Alternative Options

Consider whether a **non–macro-enabled version** or an alternative tool can meet your needs without the risks associated with running macros.

When possible, explore trusted, well-established libraries or tools rather than relying on obscure or unverified downloads from SourceForge, GitHub, or other hosting sites. Established ecosystems often provide safer, audited, and actively maintained solutions.

Practical Steps

Download cautiously: Only download from the project's official SourceForge or GitHub pages — avoid third-party mirrors or unverified sources.

Back up data: Ensure important files are backed up before opening any macro-enabled workbook to minimize risk in case of corruption or unexpected behavior.

Limit permissions: Run Excel with standard user permissions rather than administrator rights to reduce the potential impact of malicious code.

Detect Red Flags

Treat any of the following as a warning sign and proceed with extreme caution:

- Lack of documentation or unclear macro purpose.
- Requests for unusual permissions or unexpected external connections.
- Poorly rated, unverified, or inactive repositories.

- Antivirus warnings or negative community reports.

Conclusion

By combining these practices - verifying the source, reviewing documentation, inspecting code, using secure environments, and staying cautious - you can make an informed decision about trusting a macro-enabled Excel workbook from SourceForge or GitHub. When in doubt, seek guidance from a cybersecurity professional or avoid enabling macros altogether.

Privacy

I respect your privacy and keep this site as simple and transparent as possible. This page explains what information is collected, how it is used, and what is *not* collected.

No Personal Data Collected

This site does not collect, store, or process any personal information. There are no accounts, no forms, no cookies, and no tracking identifiers of any kind.

Analytics

This site uses Simple Analytics ↗ to collect basic, privacy-friendly traffic information. No personal data is collected, stored, or shared. Simple Analytics does not use cookies, does not track visitors across sites, and does not create user profiles.

The metrics ↗ information collected includes:

- the pages visited
- the referrer (if provided)
- the device type and browser (in broad categories)
- the approximate region (not precise location)
- timestamps of visits

All data is aggregated and anonymous. I use this information only to understand which pages are being used so I can keep the most relevant content up to date.

For more details, you can review the Simple Analytics approach to privacy at: https://simpleanalytics.com/privacy ↗

Third-Party Libraries

Excel to Graphviz / Relationship Visualizer incorporates third-party libraries and resources that may be distributed under licenses different from this software's own (see Credits). These licenses are generally compatible with the MIT License, though I am not a lawyer.

External Links

This site may link to external websites. I am not responsible for the content or privacy practices of those sites.

Contact

If you have questions about this privacy page or believe a required notice is missing, please contact me ↗.

Pricing

It's Absolutely Free

Excel to Graphviz / Relationship Visualizer is free. Not a free trial, not a limited tier with an upsell waiting at the end. Just free, and it always has been.

What's Included

You get the complete workbook, every worksheet, and every feature described in these docs. This includes:

- The core graphing engine that turns your Excel data into clean, meaningful Graphviz diagrams.
- The Style Designer, SQL import, JSON exchange, and SVG post-processing tools.
- More than a dozen sample workbooks that show real modeling patterns you can adapt to your own projects.
- A full PDF version of this website ↗ for offline reference.
- No account, no email signup, no license key.
- No expiration date, no feature limits, no watermarks.
- No ads on this site, and no telemetry in the tool. See the Privacy page for details.

Why It's Free

Excel to Graphviz is open source under the MIT License. It has been maintained and shared this way for more than 10 years. During that time it has reached more than 10,000

downloads and earned a SourceForge Community Choice award. The tool is useful, and there has never been a reason to put it behind a paywall.

Supporting the Project

Modern AI-powered search engines make it harder for small, independent sites like this one to be discovered. The new ranking rules tend to favor:

- Sites with many **5-star reviews**
- Sites that are **linked to by other websites, blogs, and articles**
- Projects that receive **consistent community activity**

If the tool has saved you time and you want to say thanks, here are a few ways to help keep the project visible and healthy — most of them free, and all of them appreciated.

Leave a Review

⭐ Post a review on SourceForge ↗
Positive reviews help search engines understand that the project is trusted and actively used. Even a short sentence makes a real difference.

Share the Project

🔗 **Link to the site** in your own articles, Medium posts, blogs, documentation, or internal team pages. Search engines heavily reward inbound links, and even one or two can improve visibility.

Spread the Word

📣 **Mention the tool** in forums, Q&A sites, or community discussions when it's relevant. Real-world usage signals help others discover it.

Connect with the Author

✉️ Send me a quick email ↗
I genuinely enjoy hearing how people use Relationship Visualizer in their own work. Your stories

and examples help shape future improvements and keep the project focused on real-world needs.

Contribute Feedback

🦎 Report bugs or request features on GitHub ↗

Good feedback is its own kind of thank-you and helps the tool evolve.

Donate Money

☕ Buy me a coffee ↗

Excel to Graphviz is completely free, but the project still incurs real costs — web hosting, domain fees, and the Microsoft licenses used to build and maintain the tool. Monetary contributions are entirely optional, but always appreciated.

None of this is required to use the tool, now or ever. But every small action helps keep the project discoverable in an increasingly AI-filtered web.

Credits

Thank You

The author, Jeffrey Long, extends special thanks to:

- Martin James - *Original Inspiration and mentor*
- Kyle Tuft, Roland Chee, Matthias Roth, William Lee - *Ideas/suggestions*
- Ron de Bruin - *Indispensable VBA tutorials on Excel Automation ↗, and Mac Excel Automation ↗*
- Paul Kelly - *Excel Macro Mastery ↗ YouTube videos on how to improve VBA performance*
- Arek Czak - *Polish language translations*
- Bart Broodcoorens - *Provided idea and collaboration on SQL cluster/subcluster enhancements. Provided idea, JavaScript code, and collaboration for SVG post-processing*
- Christos Samaras - *Authored VBA code to run a Windows command and return messages written to stdOut. Upon request, he kindly updated the code to also capture stdErr messages.*
- Stack Overflow - *Provided solutions to so many problems*

Third-Party Notices

Excel to Graphviz / Relationship Visualizer incorporates third-party libraries and resources that may be distributed under licenses different from this software's own. These licenses are assumed compatible with the MIT License, though I am not a lawyer.

Credit is given where credit is due

If I've unintentionally missed a required notice, please let me know ↗.

Graphviz

http://www.graphviz.org ↗

Written By: Graphviz Project Contributors ↗

Version: 3.0.0 (20220226.1711) and above

Date: 26 February 2022

License: Graphviz is free software licensed under the Eclipse Public License (EPL) v1.0 ↗

Shell and Wait

Shell and Wait ↗

Written By: Charles H. Pearson ↗

Date: 06 November 2013

License: All of the formulas and VBA code are explicitly granted to the Public Domain.

Brewer Color Schemes

Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes, Version 1.1

1. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

2. Redistributions as source code must retain the above copyright notice, this list of conditions and the following disclaimer.

   The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

   This product includes color specifications and designs developed by Cynthia Brewer (http://colorbrewer.org/ ↗).

   Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

3. The name "ColorBrewer" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact Cynthia Brewer at cbrewer@psu.edu ↗ .

4. Products derived from this software may not be called "ColorBrewer", nor may "ColorBrewer" appear in their name, without prior written permission of Cynthia Brewer.

   THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CYNTHIA BREWER, MARK HARROWER, OR THE PENNSYLVANIA STATE UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

VBA-JSON

VBA-JSON - Version: 2.3.1 ↗

Written By: Tim Hall ↗

VBA-Dictionary

VBA-Dictionary - Version: 1.4.1 ↗

Written By: Tim Hall ↗

VBA Stopwatch

VBA Stopwatch ↗

Written By: Daniel Hubman

Office RibbonX Editor

Office RibbonX Editor - Version: 1.7.1 - 23 August 2020 ↗

Written By: Fernando Andreu ↗

Rubberduck

Rubberduck - Version: 2.5.0.5244 - 21 December 2019 ↗

Written By: Mathieu Guindon ↗

Execute And Capture

ExecuteAndCapture ↗

Written By: Christos Samaras ↗

Color Palette Icon

The color palette icon used in the ribbon was sourced from clipartmax.com

Color Palette Icon - Paleta De Cores Icone @clipartmax.com ↗

Material Symbols

Most icons in the Ribbon were sourced from Material Symbols by Google ↗

Material Symbols are licensed under the Apache License, Version 2.0 ↗