

uigridlayout

Create grid layout manager

Syntax

```
g = uigridlayout
g = uigridlayout(parent)
g = uigridlayout(__,sz)
g = uigridlayout(__,Name,Value)
```

Description

`g = uigridlayout` creates a grid layout manager for an app. It positions UI components along the rows and columns of an invisible grid that spans the entire figure or a container within the figure. This command creates a 2-by-2 grid layout in a new figure and returns the `GridLayout` object. MATLAB® calls the `uifigure` function to create the figure. Use a grid layout manager if you do not want to position components by setting pixel values in `Position` vectors.

If you add components to the grid layout manager, but you do not specify the `Layout` property of the components, then the grid layout manager adds the components from left to right and then top to bottom.

`g = uigridlayout(parent)` creates a grid layout in the specified parent container. The parent can be a figure created using the `uifigure` function, or one of its child containers. [example](#)

`g = uigridlayout(__,sz)` specifies the size of the grid as the vector `sz`. The first element in this vector is the number of rows, and the second element is the number of columns. Specify the `sz` argument by itself or after the parent argument. [example](#)

`g = uigridlayout(__,Name,Value)` specifies `GridLayout` property values using one or more name-value pair arguments. Specify the name-value pair arguments after all other arguments in any of the previous syntaxes. [example](#)

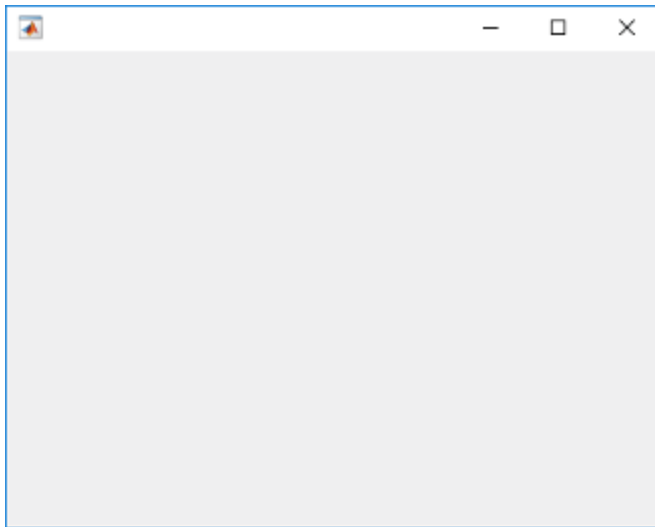
Examples

[collapse all](#)

▼ Simple 3-by-2 Grid

Create a figure and a grid. Then configure the grid by setting the row heights and column widths. In this case, configure a 3-by-2 grid in which the top two rows are fixed at 22 pixels, while the bottom row has a variable height. Set the left column to be 150 pixels wide, and set the right column to have a variable width. The grid fills the entire figure, but because the grid is invisible, the figure appears to be empty.

```
fig = uifigure('Position',[100 100 440 320]);
g = uigridlayout(fig);
g.RowHeight = {22,22,'1x'};
g.ColumnWidth = {150,'1x'};
```



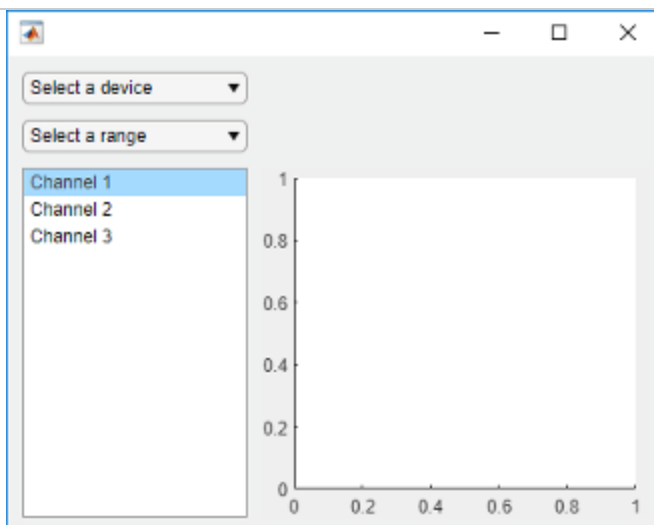
Add two drop-downs, a list box, and an axes component to the grid. If you do not specify row or column locations for the Layout property of components, they populate the grid from left to right and top to bottom by default. In this case, move the second drop-down (dd2) and the list box (chanlist) to specific rows and columns by setting the Layout property.

```
% Device drop-down
dd1 = uidropdown(g);
dd1.Items = {'Select a device'};

% Range drop-down
dd2 = uidropdown(g);
dd2.Items = {'Select a range'};
dd2.Layout.Row = 2;
dd2.Layout.Column = 1;

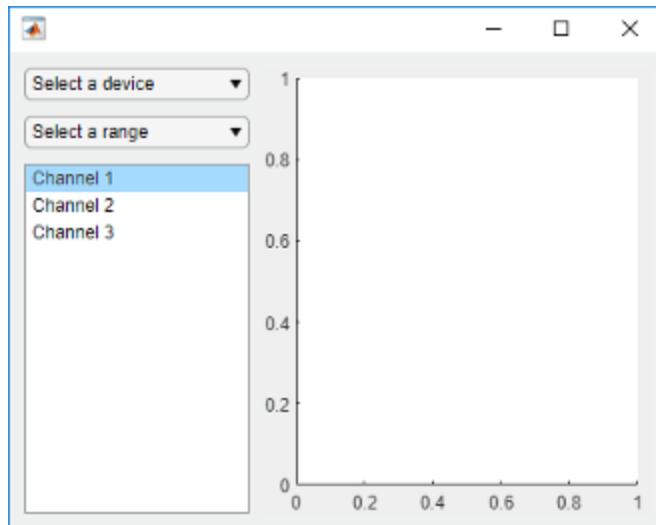
% List box
chanlist = uilistbox(g);
chanlist.Items = {'Channel 1','Channel 2','Channel 3'};
chanlist.Layout.Row = 3;
chanlist.Layout.Column = 1;

% Axes
ax = uiaxes(g);
```

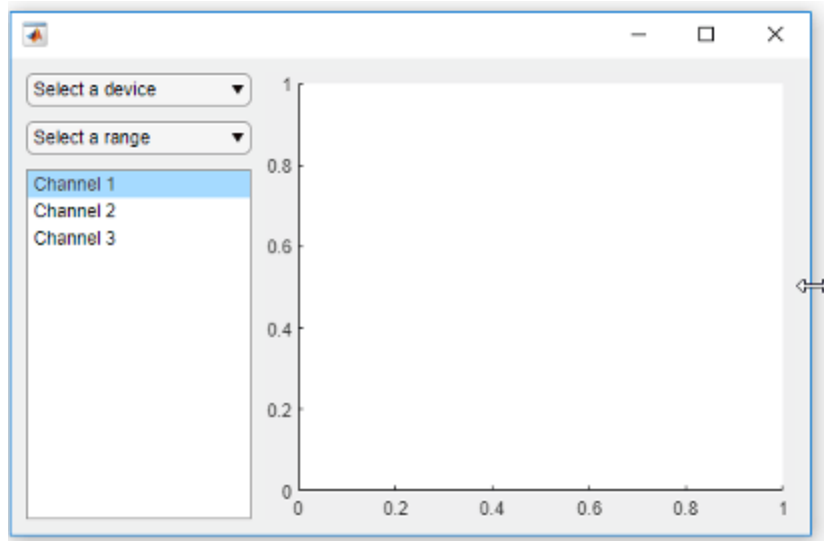


Set the axes to span rows 1 through 3, filling the vertical space.

```
ax.Layout.Row = [1 3];
```



When you resize the figure, the axes grow and shrink to fill the available space that the left column does not use. The components on the left side remain fixed because that column is defined as 150 pixels wide.

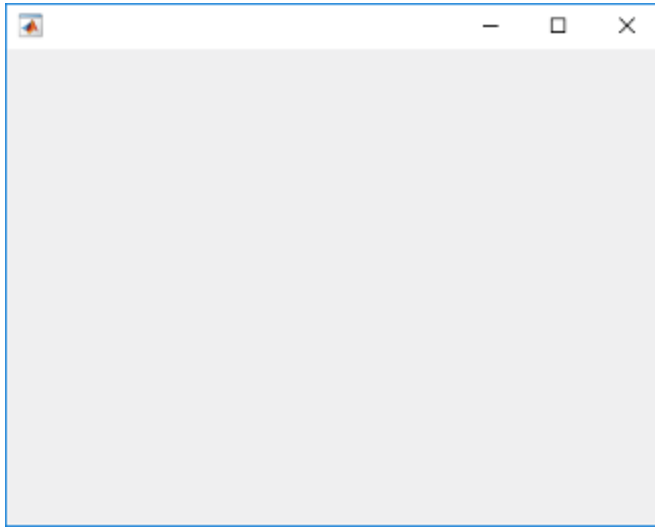


✓ Variable-Width Columns That Have Different Weights

When you assign different weights to variable-width columns, the width of each column is proportional to the weight. Variable-height rows allocate space the same way.

Create a figure and a 2-by-2 grid. Then configure the row heights and the column widths. In this case, the top row is fixed at 40 pixels, while the bottom row has a variable height. Both columns have a variable width, but the second column has twice the weight of the first column. Because the grid is invisible, the figure appears to be empty.

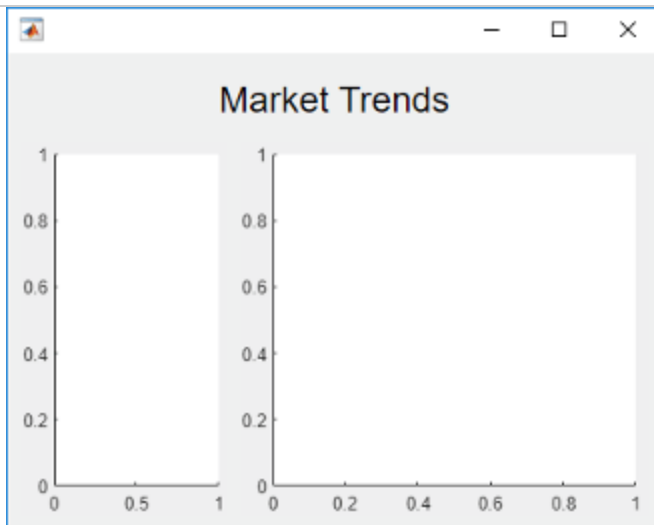
```
fig = uifigure('Position',[100 100 437 317]);  
g = uigridlayout(fig,[2 2]);  
g.RowHeight = {40,'1x'};  
g.ColumnWidth = {'1x','2x'};
```



Add a label across both columns of the top row. Then add an axes component to each column of the bottom row.

```
% Add title
title = uilabel(g,'Text','Market Trends');
title.HorizontalAlignment = 'center';
title.FontSize = 24;
title.Layout.Row = 1;
title.Layout.Column = [1,2];

% Add two axes
ax1 = uiaxes(g);
ax2 = uiaxes(g);
```



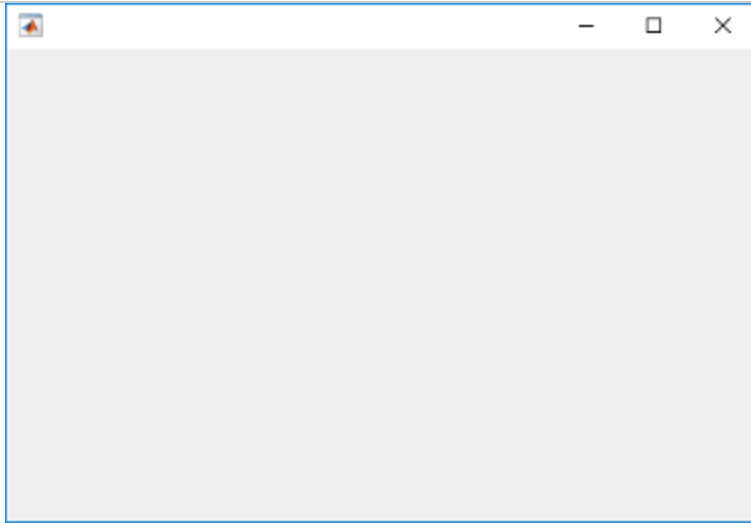
Because the left column has a weight of 1, and the right column has a weight of 2, the axes on the right use twice as much of the horizontal space as the axes on the left. The grid maintains this ratio when you resize the figure.

▼ Figure and Panel That Each Have a Grid

One way to arrange UI components in a panel is to create a grid in the panel.

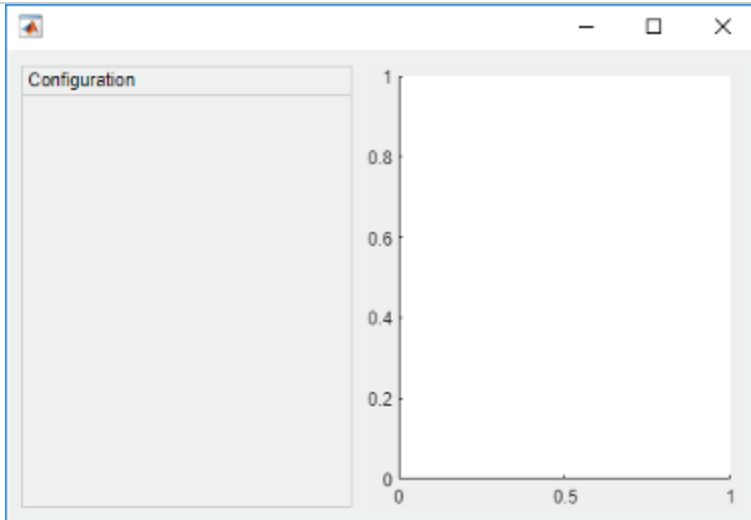
Create a figure and a 1-by-2 grid. Then configure the column widths. In this case, the left column is fixed at 220 pixels, while the right column has a variable width. Because the grid is invisible, the figure appears to be empty.

```
fig = uifigure('Position',[100 100 500 315]);  
grid1 = uigridlayout(fig,[1 2]);  
grid1.ColumnWidth = {220,'1x'};
```



Add a panel and axes to grid1.

```
p = uipanel(grid1,'Title','Configuration');  
ax = uiaxes(grid1);
```



Create a grid called `grid2` inside the panel. Then place three components and three labels inside `grid2`. The left column of `grid2` aligns the labels with the components. Since each row uses only 22 pixels, the panel has extra space below the third row.

```

% Grid in the panel
grid2 = uigridlayout(p,[3 2]);
grid2.RowHeight = {22,22,22};
grid2.ColumnWidth = {80,'1x'};

% Device label
dlabel = uilabel(grid2);
dlabel.HorizontalAlignment = 'right';
dlabel.Text = 'Device';

% Device drop-down
devicedd = uidropdown(grid2);
devicedd.Items = {'Select a device'};

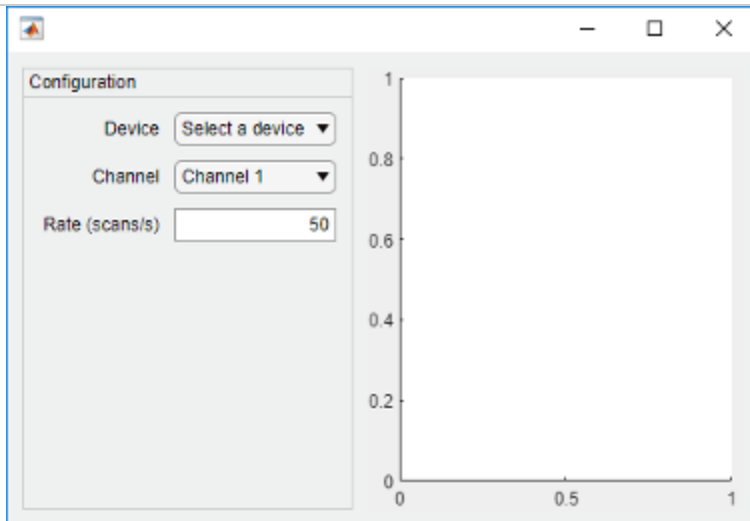
% Channel label
chlabel = uilabel(grid2);
chlabel.HorizontalAlignment = 'right';
chlabel.Text = 'Channel';

% Channel drop-down
channeldd = uidropdown(grid2);
channeldd.Items = {'Channel 1', 'Channel 2'};

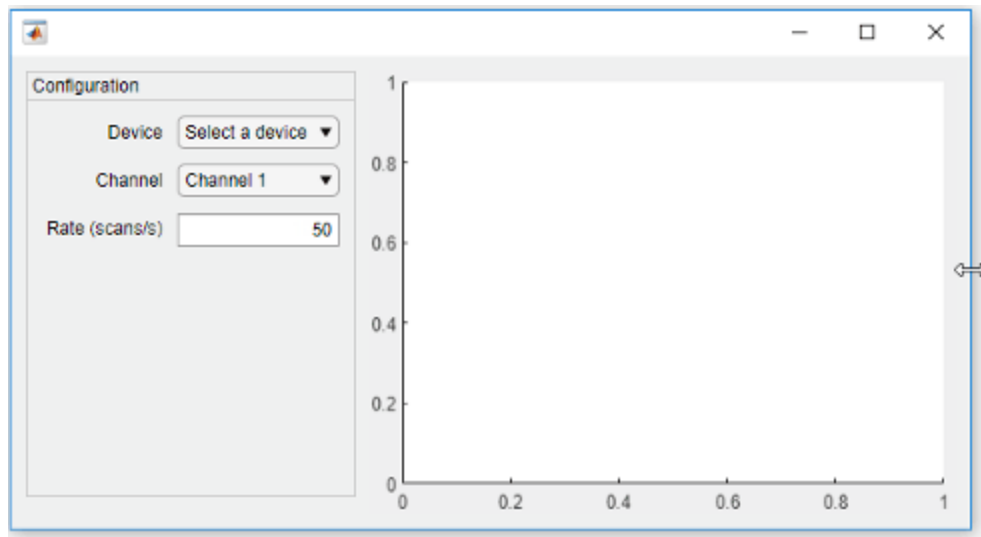
% Rate Label
ratelabel = uilabel(grid2);
ratelabel.HorizontalAlignment = 'right';
ratelabel.Text = 'Rate (scans/s)';

% Rate edit field
ef = uieditfield(grid2, 'numeric');
ef.Value = 50;

```



When you resize the figure, the axes grow and shrink to fill the available space that the left column of grid1 does not use. The panel remains fixed because that column is defined as 220 pixels wide.

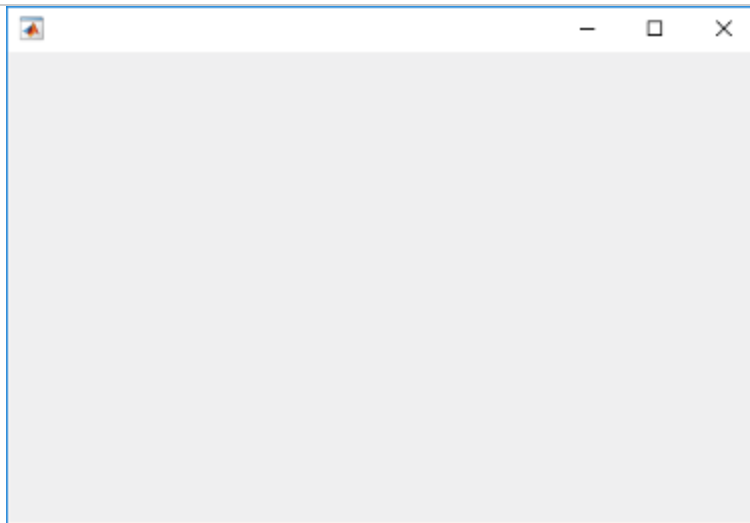


▼ Nested Grid

Nested grids allow you to manage subsets of components. In this case, there are three grids: one grid that covers the entire figure, a second grid that manages a vertical stack of components, and a third grid that manages two buttons that are side by side at the bottom of the vertical stack.

Create a figure and a 1-by-2 grid. Then configure the column widths. In this case, the left column is fixed at 150 pixels, while the right column has a variable width. Because the grid is invisible, the figure appears to be empty.

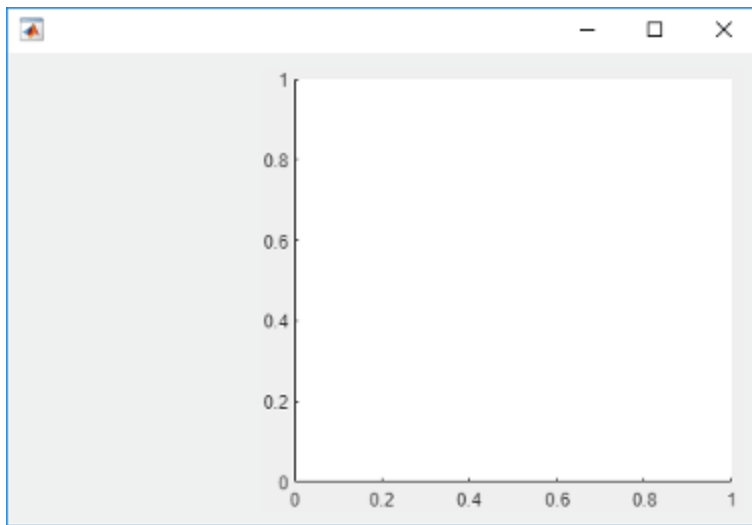
```
fig = uifigure('Position',[100 100 500 315]);
grid1 = uigridlayout(fig,[1 2]);
grid1.ColumnWidth = {150,'1x'};
```



Create a 4-by-1 grid called `grid2` inside the first column of `grid1`. Then add an axes component to the second column of `grid1`.

```
grid2 = uigridlayout(grid1,[4 1]);
grid2.RowHeight = {22,22,22,44};

ax = uiaxes(grid1);
```

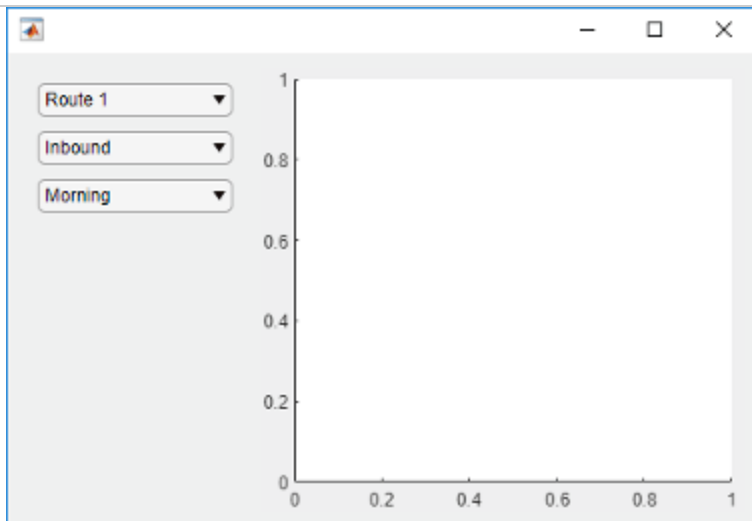


Add three drop-downs to the first three rows of grid2.

```
% Routes drop-down
ddRoutes = uiddropdown(grid2);
ddRoutes.Items = {'Route 1', 'Route 2'};

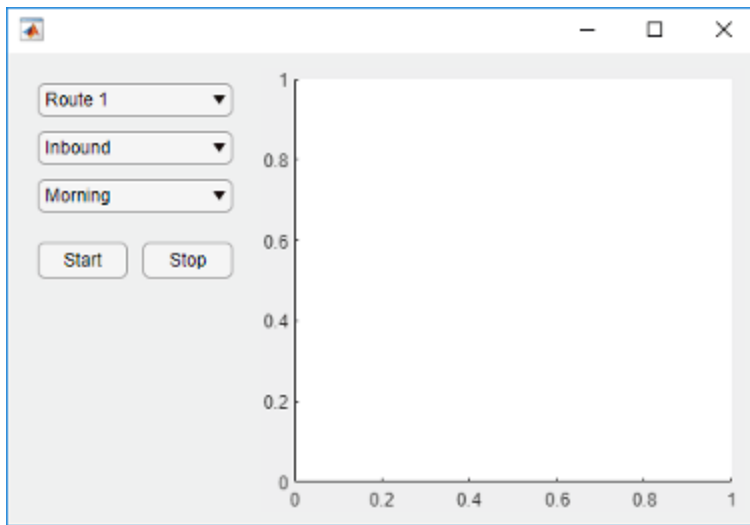
% Direction drop-down
ddDirection = uiddropdown(grid2);
ddDirection.Items = {'Inbound', 'Outbound'};

% Time drop-down
ddTime = uiddropdown(grid2);
ddTime.Items = {'Morning', 'Afternoon'};
```

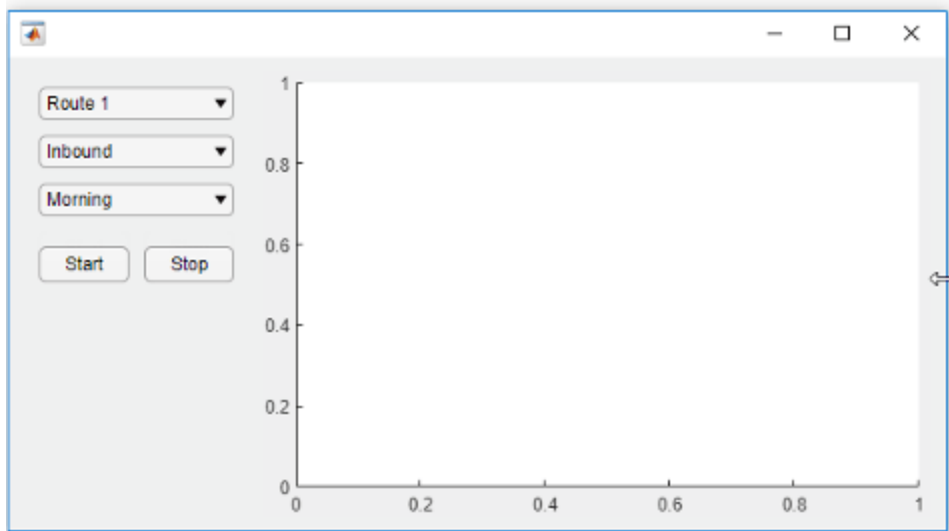


Create a 1-by-2 grid called grid3 inside the last row of grid2. Then add two buttons to grid3. Remove the padding on the left and right edges of grid3 so that the buttons align with the left and right edges of the drop-downs.

```
grid3 = uigridlayout(grid2,[1 2]);
grid3.Padding = [0 10 0 10];
b1 = uibutton(grid3,'Text','Start');
b2 = uibutton(grid3,'Text','Stop');
```

When you resize the figure, the axes grow and shrink to fill the available space that the left column of `grid1` does not use. The components on the left side remain fixed because that column is defined as 150 pixels wide.



▼ Scroll to Location in Grid

Bring child components of a scrollable grid layout into view by specifying pixel coordinates or a location name.

Create a 5-by-2 grid layout and set the `Scrollable` property of the grid to 'on'. Then add a label, a table, and a panel to the grid. Set the `Scrollable` property of the panel to 'off' and then add a chart to the panel.

```

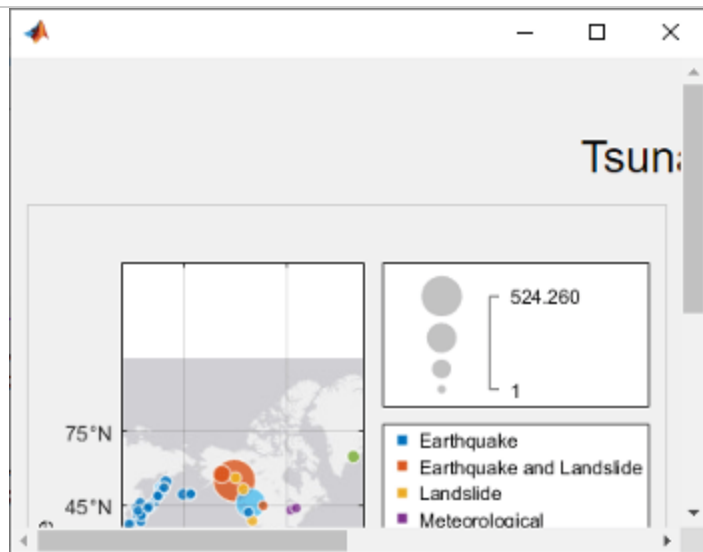
fig = uifigure('Position',[782 497 435 311]);
g = uigridlayout(fig,'Scrollable','on');
g.RowHeight = {22,40,22,22,400};
g.ColumnWidth = {400,400};

lbl = uilabel(g,'Text','Tsunamis');
lbl.Layout.Row = 2;
lbl.Layout.Column = [1,2];
lbl.HorizontalAlignment = 'center';
lbl.FontSize = 28;

tsunamis = readtable('tsunamis.xlsx');
tsunamis.Cause = categorical(tsunamis.Cause);
t = uitable(g,'Data',tsunamis);
t.Layout.Row = [3,5];
t.Layout.Column = 2;

p = uipanel(g);
p.Scrollable = 'off';
p.Layout.Row = [3,5];
p.Layout.Column = 1;
gb = geobubble(p,tsunamis.Latitude,tsunamis.Longitude,...
    tsunamis.MaxHeight,tsunamis.Cause);

```

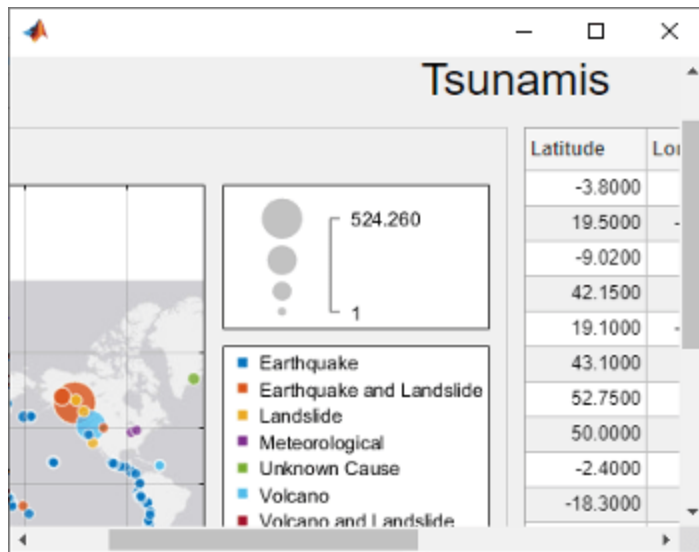


Scroll to a location in the grid.

```

scroll(g,100,-30)

```



Now use location names to scroll to the bottom-right corner of the grid.

```
scroll(g,'bottom','right')
```

52.7500	159.5000	1952	11	4
50.0000	156.5000	1953	3	18
-2.4000	147.4000	1953	6	27
-18.3000	178.2000	1953	9	14
34.0000	141.7000	1953	11	25
41.7000	-86.8830	1954	6	26
67.8000	-32.2830	1954	10	NaN
-30.0000	-72.0000	1955	4	19
39.5000	23.0000	1956	11	2
51.2920	-175.6290	1957	3	9
-1.5000	137.0000	1957	6	22
16.5000	-99.0000	1957	7	28

✓ Create Grid That Automatically Adjusts to Fit Content

Create a grid layout manager that automatically adjusts its row and column sizes to fit components like labels as you add them to your app, or when the font size of text-based components changes dynamically.

Create a 5-by-2 grid layout manager. Specify 'fit' for the first column width and the first four row heights.

```
fig = uifigure('Position',[500 500 430 310]);

g = uigridlayout(fig);
g.ColumnWidth = {'fit',120,'1x'};
g.RowHeight = {'fit','fit','fit','fit','1x'};
```

Create a title label that spans all the columns. Then, create labels and controls along the rows of the first and second columns. Lastly, create a table UI component in the last column that spans rows two through five. Notice that as you add components to the rows and columns with 'fit' height and width, the size of those row and columns automatically adjusts to fit the new content.

```

patientLabel = uilabel(g,'Text','Patient:');
patientLabel.Layout.Row = 2;
patientLabel.Layout.Column = 1;
patientEdit = uieditfield(g);

ageLabel = uilabel(g,'Text','Age:');
ageLabel.Layout.Row = 3;
ageLabel.Layout.Column = 1;
ageEdit = uidropdown(g,'Items',{'<20','20 - 40','40 - 60','>60'});

stateLabel = uilabel(g,'Text','State:');
stateLabel.VerticalAlignment = 'top';
stateLabel.Layout.Row = 4;
stateLabel.Layout.Column = 1;
stateLB = uilistbox(g,'Items',{'MA','VA','WY'});

tdata = readtable('patients.dat');
uit = uitable(g);
uit.Data = tdata;
uit.Layout.Row = [2 5];
uit.Layout.Column = 3;

```

LastName	Gender	Age
Smith	Male	
Johnson	Male	
Williams	Female	
Jones	Female	
Brown	Female	
Davis	Female	
Miller	Female	
Wilson	Male	
Moore	Male	
Taylor	Female	
Anderson	Female	

Now, change the font name and font size of all the UI components in the grid.

```

set(allchild(g),'FontName','Lucida Console','FontSize',20)

```

LastName	Gender
Smith	Male
Johns...	Male
Willi...	Fema...
Jones	Fema...
Brown	Fema...
Davis	Fema...
Miller	Fema...
Wilson	Male

▼ Hide Rows Based on Run-Time Conditions

Hide components within a row of a grid based on the user's selection in a drop-down menu.

Create a program file called `showhide.m`. In the file,

- Create a 1-by-2 grid in the figure and call it `grid1`. Use this grid to manage a panel and an axes component.
- Create a 3-by-2 grid inside the panel and call it `grid2`. Use this grid to manage the layout of a drop-down menu, two spinners, and their labels.
- Create a callback function called `findMethodSelected` for the drop-down menu. When the value of the drop-down menu changes to 'Quartiles', the callback hides the components in a second row of `grid2` by setting `grid2.RowHeight{2}` to 0.

Then, run the file.

```

function showhide

    f = uifigure('Name','Statistical Analysis');

    % Create grid1 in the figure
    grid1 = uigridlayout(f);
    grid1.RowHeight = {'1x'};
    grid1.ColumnWidth= {220,'1x'};

    % Add a panel and axes
    p = uipanel(grid1);
    ax = uiaxes(grid1);

    % Create grid2 in the panel
    grid2 = uigridlayout(p);
    grid2.RowHeight = {22, 22, 22};
    grid2.ColumnWidth = {80,'1x'};

    % Add method label and drop-down
    findMethodLabel = uilabel(grid2,'Text','Find Method:');
    findMethod = uidropdown(grid2);
    findMethod.Items = {'Moving median','Quartiles'};
    findMethod.ValueChangedFcn = @findMethodSelected;

    % Add window size label and spinner
    winSizeLabel = uilabel(grid2,'Text','Window Size:');
    winSize = uispinner(grid2,'Value',0);

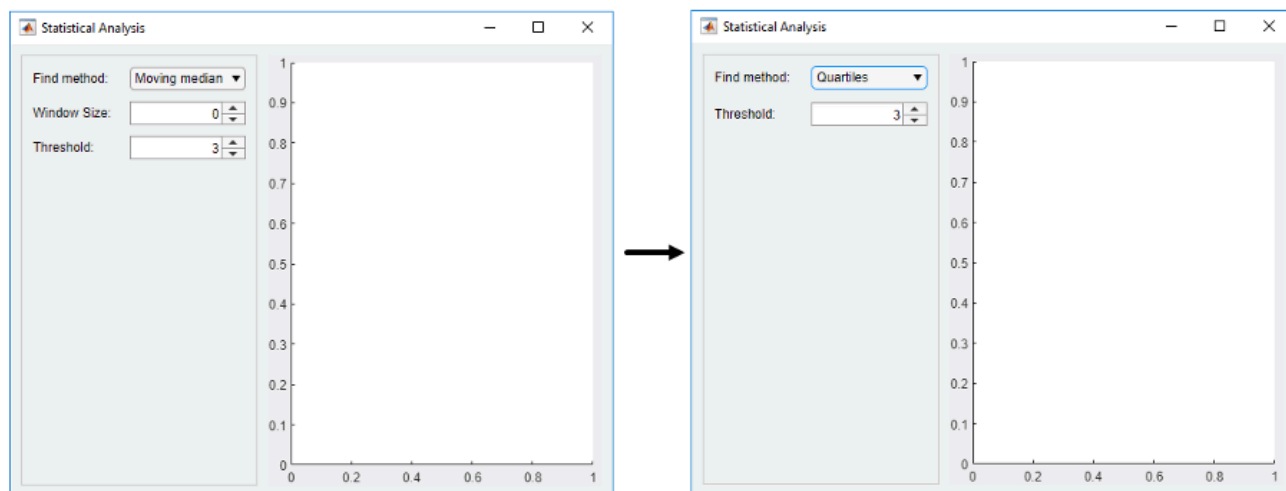
    % Add threshold label and spinner
    thresLabel = uilabel(grid2,'Text','Threshold:');
    thres = uispinner(grid2,'Value',3);

    function findMethodSelected(src,~)
        method = src.Value;

        switch method
        case 'Quartiles'
            % Collapse the second row (hides winSize spinner)
            grid2.RowHeight{2} = 0;
        case 'Moving median'
            % Expand the second row
            grid2.RowHeight{2} = 22;
        end
    end
end
end

```

When you set the **Find Method** to **Quartiles** in the app, the **Window Size** label and the spinner next to it become hidden.



Input Arguments

collapse all



parent – Parent container

Figure object (default) | Tab object | Panel object | ButtonGroup object | GridLayout object

Parent container, specified as a Figure object created using the `uifigure` function or one of its child containers: Tab, Panel, ButtonGroup, or GridLayout. If you do not specify a parent container, MATLAB calls the `uifigure` function to create a new Figure object that serves as the parent container.



sz – Size

[2 2] (default) | two-element vector

Size of the grid, specified as a two-element vector. The first element is the number of rows, and the second element is the number of columns. Both values must be greater than 0. When you specify this argument without setting the `RowHeight` or `ColumnWidth` properties, MATLAB automatically sets the row heights and column widths to '1x'.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name in quotes.

Example: `uigridlayout('RowHeight',{100,100})` creates a grid containing two rows that are 100 pixels in height.



Note

The properties listed here are only a subset. For a complete list, see `GridLayout`.



ColumnWidth – Column width

{'1x', '1x'} (default) | cell array | string array | numeric array

Column width, specified as a cell array containing either 'fit', numbers, or numbers paired with 'x' characters. You can specify any combination of values. The number of elements in the cell array controls the number of columns in the grid. For example, to create a 4-column grid, specify a 1-by-4 cell array. Column width can be specified as a string array or numeric array, only if the elements specified are of the same type, like ["1x" "2x" "1x"] or [100 200 50].

There are three different types of column widths:

- Fit width – Specify 'fit'. Column width automatically adjusts to fit its contents. For text-based components, 'fit' width adjusts with font properties to show the whole text. For non text-based components, 'fit' width is based on the default size of the component and other factors. Use 'fit' width if you want to avoid hard-coding the column width to fit components, or if your app is translated to another language or runs on different platforms.
- Fixed width in pixels – Specify a number. The column width is fixed at the number of pixels you specify. When the parent container resizes, the column width does not change.
- Variable width – Specify a number paired with an 'x' character (for example, '1x'). When the parent container resizes, the column width grows or shrinks. Variable-width columns fill the remaining horizontal space that the fixed-width columns do not use. The number you pair with the 'x' character is a weight for dividing up the remaining space among all the variable-width columns. If the grid has only one variable-width column, then it uses all the remaining space regardless of the number. If there are multiple variable-width columns that use the same number, then they share the space equally. Otherwise, the amount of space is proportional to the number.

For example, {'fit', 200, '2x', '1x'} specifies that the width of the first column is sized to fit its content, the second column is fixed at 200 pixels, and the last two columns share the remaining horizontal space. The third column uses twice as much space as the fourth column.

Changing certain aspects of a layout can affect the value of this property. For example, adding more components to a fully populated grid changes the size of the grid to accommodate the new components.

Changing the ColumnWidth property on a grid layout that already contains components does not change the layout of the components. For example, if you try to dynamically delete a column that contains components, the ColumnWidth property does not change until you move those components out of that column.



RowHeight – Row height

{'1x', '1x'} (default) | cell array | string array | numeric array

Row height, specified as a cell array containing either 'fit', numbers, or numbers paired with 'x' characters. You can specify any combination of values. The number of elements in the cell array controls the number of rows in the grid. For example, to create a grid that has 4 rows, specify a 1-by-4 cell array. Row height can be specified as a string array or numeric array, only if the elements specified are of the same type, like ["1x" "2x" "1x"] or [100 200 50].

There are three different types of row heights:

- Fit height – Specify 'fit'. Row height automatically adjusts to fit its contents. For text-based components, 'fit' height adjusts with font properties to show the whole text. For non text-based components, 'fit' height is based on the default size of the component and other factors. Use 'fit' height if you want to avoid

hard-coding the row height to fit components, or if your app is translated to another language or runs on different platforms.

- Fixed height in pixels — Specify a number. The row height is fixed at the number of pixels you specify. When the parent container resizes, the row height does not change.
- Variable height — Specify a number paired with an 'x' character (for example, '1x'). When the parent container resizes, the row grows or shrinks. Variable-height rows fill the remaining vertical space that the fixed-height rows do not use. The number you pair with the 'x' character is a weight for dividing up the remaining space among all the variable-height rows. If the grid has only one variable-height row, then it uses all the remaining space regardless of the number. If there are multiple variable-height rows that use the same number, then they share the space equally. Otherwise, the amount of space is proportional to the number.

For example, `{'fit', 200, '2x', '1x'}` specifies that the height of the first row is sized to fit its content, the second row is fixed at 200 pixels, and the last two rows share the remaining vertical space. The third row uses twice as much space as the fourth row.

Changing certain aspects of a layout can affect the value of this property. For example, adding more components to a fully populated grid changes the size of the grid to accommodate the new components.

Changing the `RowHeight` property on a grid layout that already contains components does not change the layout of the components. For example, if you try to dynamically delete a row that contains components, the `RowHeight` property does not change until you move those components out of that row.

▼ **Position — Location and size of grid layout manager**
[left bottom width height]

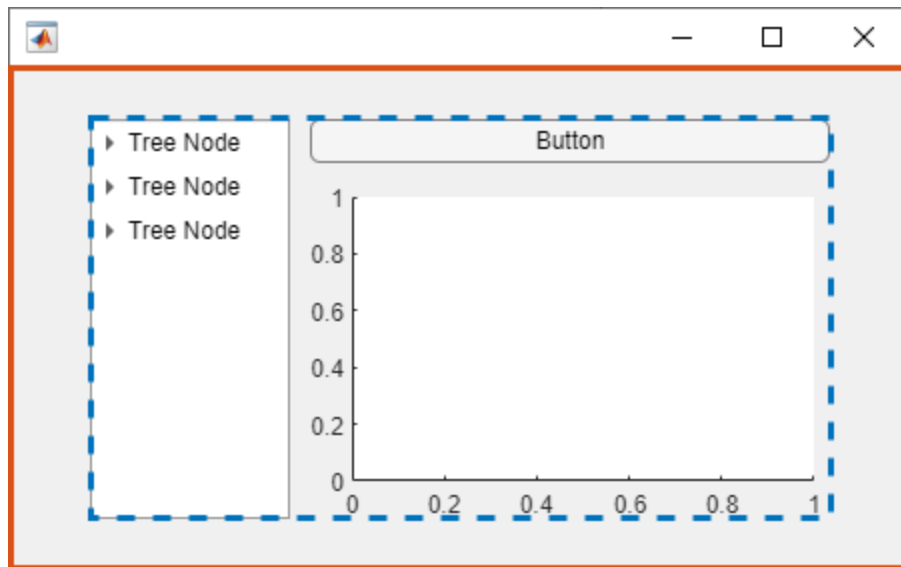
This property is read-only.

Location and size of the grid layout manager, returned as a four-element vector of the form [left bottom width height]. This table describes each element in the vector.

Element	Description
left	Distance from the inner left edge of the parent container to the left edge of the grid layout manager
bottom	Distance from the inner bottom edge of the parent container to the bottom edge of the grid layout manager
width	Distance between the left and right edges of the grid layout manager
height	Distance between the bottom and top edges of the grid layout manager

All measurements are in pixel units.

This image shows the areas defined by the `Position` value (orange solid line) and the `InnerPosition` value (blue dashed line) of a grid layout manager with some UI components.



Tips

To view the list of component objects in the grid, query the `Children` property of the grid. Changing the order in the list does not change the layout in the grid.

Version History

Introduced in R2018b

[expand all](#)

- > **R2022b:** Access grid layout manager location and size
- > **R2022a:** Table, list box, and image UI components resize to fit content ⚠
- > **R2020b:** GridLayout background is no longer transparent ⚠

See Also

Functions

`scroll|uifigure`

Properties

`GridLayout`

Tools

App Designer

Topics

Use Grid Layout Managers in App Designer

YOU MIGHT ALSO BE INTERESTED IN

Parsing Text with Patterns

Search for and modify text using patterns.

[Learn more](#)