

Embedded Components

▼ Tables as Embedded Components

▼ Table Properties

In Maple 2016 some [Table](#) properties can be accessed programmatically, meaning properties can be updated using the [DocumentTools](#) package. To enable this, Tables now have a name (or identity), which is listed in the [Table Properties](#) dialogue. Note the name is not the same as the title.



This Table is a Dynamic Component

The following commands were used to update the Table above.

```
with(DocumentTools) :
```

```
SetProperty("Table0", 'title', "This Table is a Dynamic Component");
```

```
SetProperty("Table0", 'fillcolor'[1, 1], blue);
```

```
SetProperty("Table0", 'fillcolor'[2, 1..2], yellow);
```

```
SetProperty("Table0", 'exterior', none);
```

```
SetProperty("Table0", 'interior', none);
```

```
SetProperty("Table0", 'visible'[3,..], false );
```

```
SetProperty("Table0", 'width', 400);
```

```
SetProperty("Table0", 'alignment', 'center');
```

▼ Programmable Tables

Tables can be also be created and inserted (or *embedded*) using **DocumentTools** commands:

with(DocumentTools) :

with(DocumentTools:-Layout) :

```
InsertContent( Worksheet( Table( Column( )$3,
                               Row( Cell("x"), Cell("y"), Cell("z") ), Row( Cell(1), Cell(2), Cell(3) ),
                               ))):
```

x	y	z
1	2	3

The **Tabulate** command provides built-in support for laying out **lists**, **Arrays**, **Vectors**, **Matrices** and **DataFrames** containing strings, numbers, math expressions, and plots. By constructing the content using the **DocumentTools** commands, further customization is possible and the inserted Table can also be subsequently altered programmatically.

> $M := \text{ImportMatrix}(\text{cat}(\text{kernelopts}(\text{datadir}), \text{"datasets/sunspots.csv"})) [1..15,..]$

$$M := \left[\begin{array}{l} 15 \times 2 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran_order} \end{array} \right]$$

> $(m, n) := \text{upperbound}(M)$

$m, n := 15, 2$

```
> lookup := InsertContent(
Worksheet(
Table('identity'="sunspot_table", 'alignment=center', 'interior'=none, 'width'=300,
'widthmode'=pixels, Column( )$n,
seq(
Row(
seq(
```

```

Cell(
  `fillcolor` = `if`(row = 1, blue, `if`(row mod 2 = 1, "#ccddff", white)),
  `if`(row = 1, Textfield(Font(M[row, col], color = white, bold)), sprintf("%a",
M[row, col]))
),
col = 1 ..n
),
row = 1 ..m )
), 'output' = table
):

```

Date	Mean Sunspot Number
1700	5.0
1701	11.0
1702	16.0
1703	23.0
1704	36.0
1705	58.0
1706	29.0
1707	20.0
1708	10.0
1709	8.0
1710	3.0
1711	0.
1712	0.
1713	2.0

The Table inserted above may receive a new identity upon insertion if the value of the **identity** option supplied to the Table constructor is already in use in the worksheet. The name **lookup** has been assigned a [table](#) which can be used to look up the identity that has been used for insertion. In this example, the identity has not changed since the supplied identity is not already used in any other component in the worksheet.

```
> lookup["sunspot_table"]  
"sunspot_table"
```

The next command hides rows 11 through 14 of the above Table. The identity of the inserted Table is used for this.

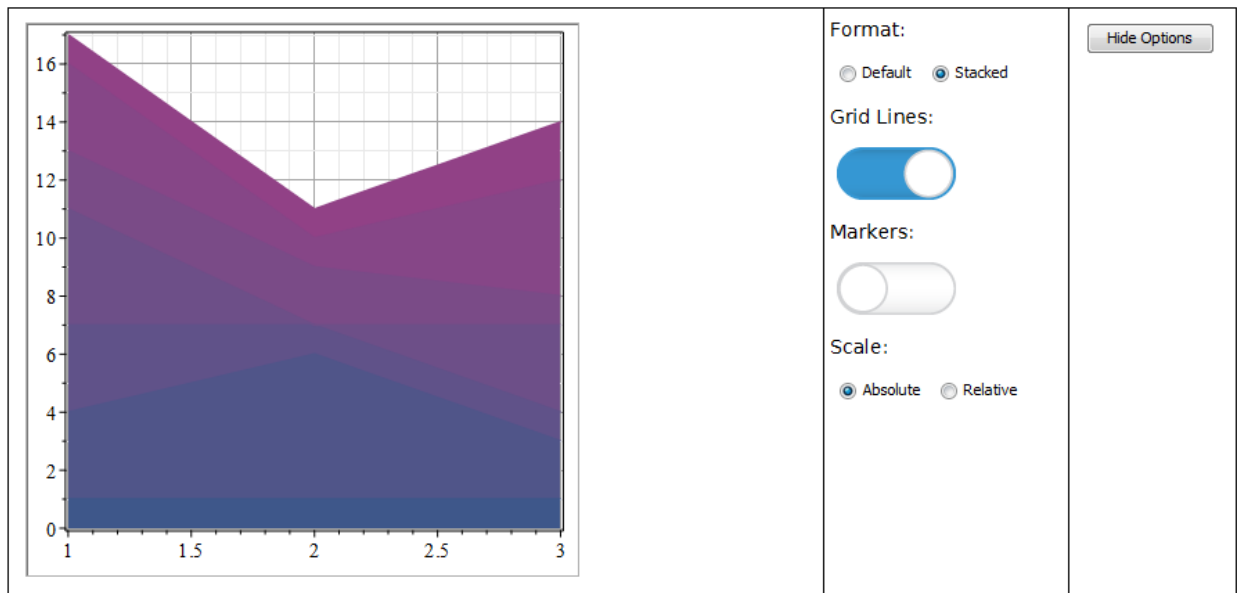
```
> SetProperty(lookup["sunspot_table"], visible[11..14,..], false)
```

And those rows can be made visible once more.

```
> SetProperty(lookup["sunspot_table"], visible[11..14,..], true)
```

▼ Hidden Options Example

In this example, we use the table's ability to show or hide cells, rows, or columns. This example presents an interactive application of an [area chart](#) in which you can easily change some options that affect how the plot is displayed. After you get the look you want, you can hide the options in order to reduce screen clutter. For demonstration purposes, the "Hide Options" button is implemented using a captioned **Button** component, but this could be reduced to a simple unobtrusive icon.



The above application was created by dragging components off the **Components palette**. The action handlers of the various embedded components obtain the relevant options and generate the plot using the **AreaChartPlot** module defined in the following Code Edit Region. Note how **AreaChartPlot:-Options** implements hiding the options column.



```
AreaChartPlot := module()
```

▼ Plot Component

The following properties of a **Plot Component** can now be accessed programmatically:

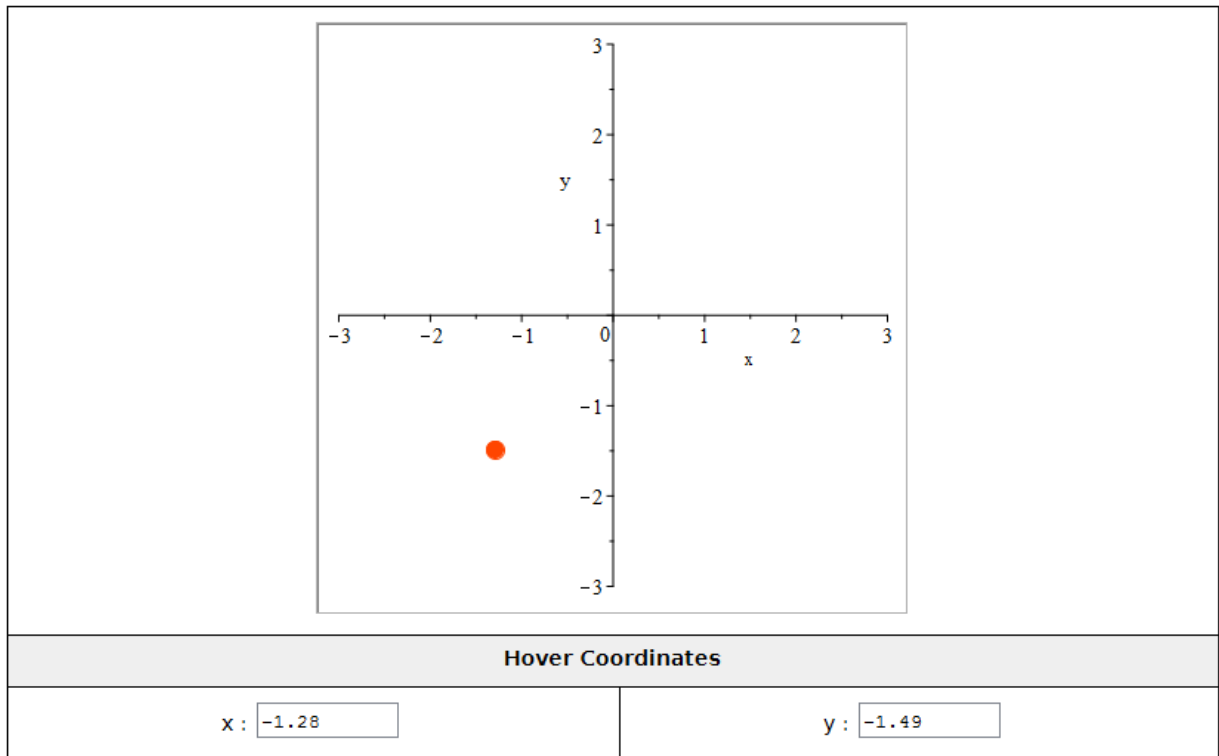
- **hoverx**: the x-coordinate position where the mouse is currently hovering
- **hovery**: the y-coordinate position where the mouse is currently hovering

In addition, it is now possible for the **Plot** component to run action code on the event of a mouse hover.

▼ Example 1

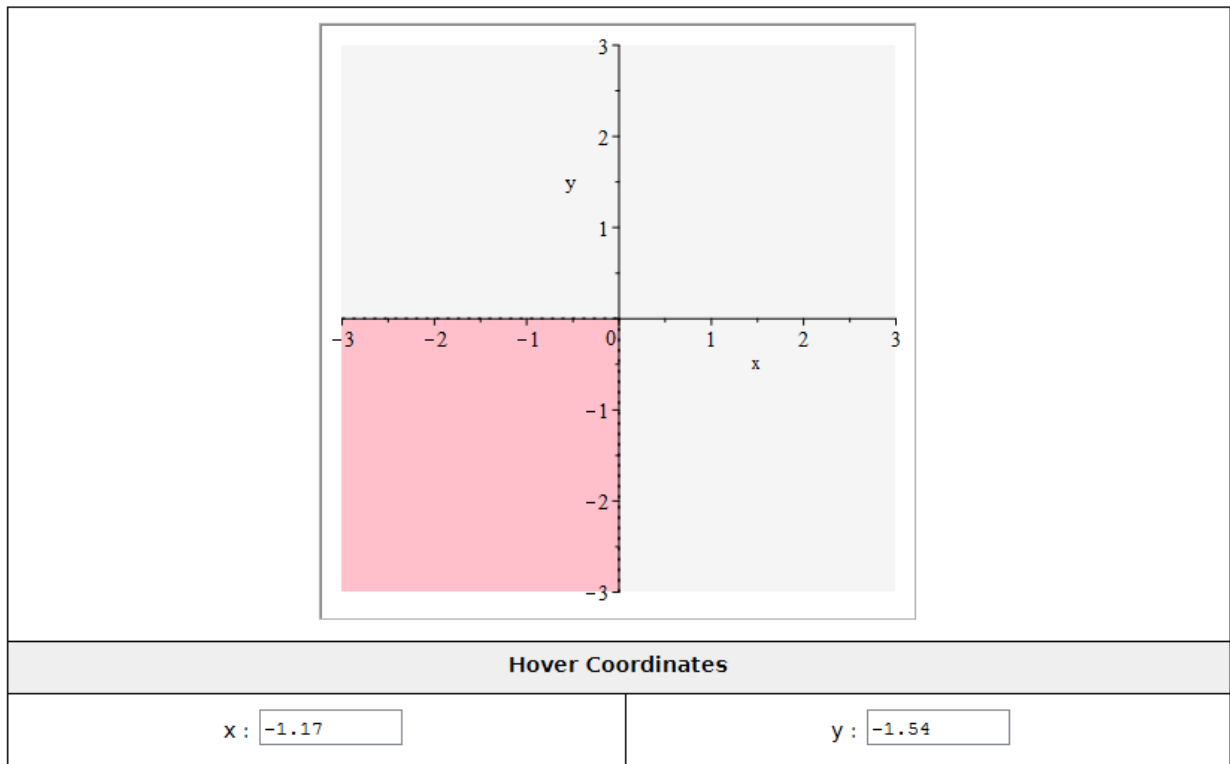
The hover over action on **Plot** components allows code to be run any time the mouse

hovers in the plot. In this example, the hover action code updates two `TextArea` components with information on the current position of the mouse cursor. In addition, the action code updates the plot shown in the `Plot` component with a `pointplot` corresponding to the current position of the mouse.



▼ Example 2

In this example, as the mouse moves from one quadrant of the plot to the next, the action code updates the plot shown in the `Plot` component. The action code also updates the `TextArea` components.

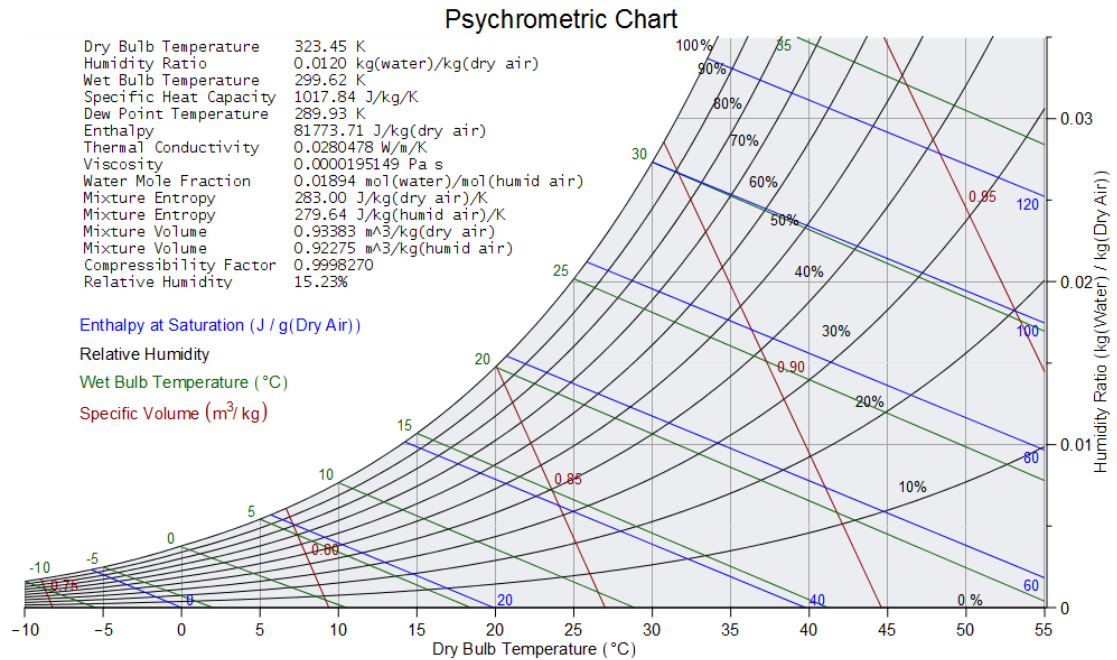


▼ Example 3

As a part of the new [ThermophysicalData](#) package, you can now generate [psychrometric charts](#) for humid air. The following chart, for example, is generated at 101325 Pa.

In this example, as the mouse hovers over the psychrometric chart, Maple:

- Calculates thermophysical data for the current dry bulb temperature and humidity ratio.
- Overlays the information on top of the chart.



▼ Math Containers

The following properties of a [Math Container](#) can now be accessed programmatically:

- **editable**: specify if the math container contents can be edited or not.
- **minimumpixelheight**: set the minimum pixel height. Used in conjunction with the **autofit** option.
- **minimumpixelwidth**: set the minimum pixel width. Used in conjunction with the **autofit** option.

Note: These properties can also be accessed through the **Math Container Properties** window.

▼ Component Font Color



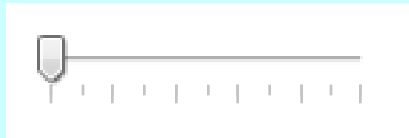
You can now control the font color for Buttons, Check Boxes, Combo Boxes, Labels, List Boxes, Radio Buttons, Sliders, and Text Areas. This can be adjusted from the component properties dialog or programmatically accessed using the following option:

- **fontcolor**: specify the font color of the component.

▼ Component Transparency and Fill Color

By default, most embedded components are now transparent, meaning that they will use the same background color as the table cell they are in. Check Boxes, Labels, List Boxes, Math Expression Component, Plot Components, Radio Buttons, Sliders, and Text Areas are not transparent by default but now have a new customizable background **fillcolor** option. This can be adjusted from the component properties dialog or programmatically accessed using the following option:

- **fillcolor**: specify the background color of the component.

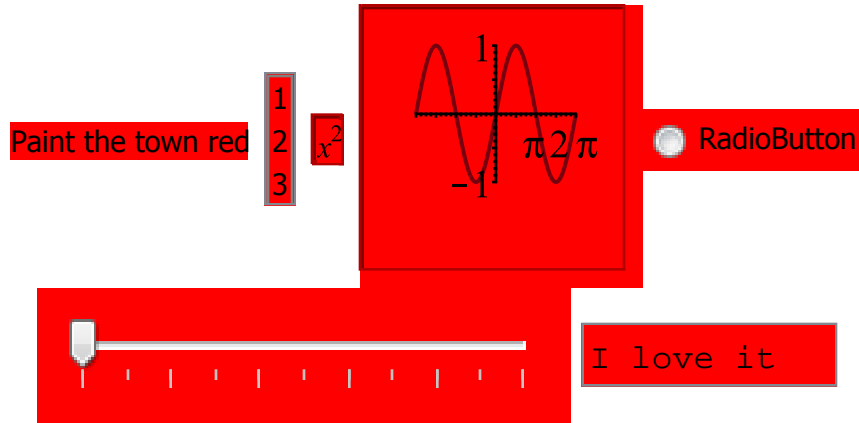
<p>Examples: Transparent Components</p>	 <p>Shortcut</p>	
<p>Examples: Components with fillcolor option</p>	<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;">background</div> <p>> <i>DocumentTools:-</i> <i>SetProperty</i>("TextAreaCT1", <i>fillcolor</i>, "WhiteSmoke")</p>	 <p>> <i>DocumentTools:-</i> <i>SetProperty</i>("SliderCT1", <i>fillcolor</i>, [128, 128, 128])</p>

▼ Example: Background Colors on Components

The following components can make use of the background fill color option.

```
> with(DocumentTools) : with(DocumentTools:-Layout) : with(DocumentTools:-
  Components) :
  InsertContent(
  Worksheet(
  Table( 'exterior'='none', Column( ),
  Row( Cell(
    Label("Paint the town red",'fillcolor'="#ff0000"),
    ListBox( ["1", "2", "3"],'fillcolor'="#ff0000"),
    MathContainer(x^2,'fillcolor'="#ff0000','autofit'),
    Plot(plot(sin),'fillcolor'="#ff0000','pixelwidth'= 100,'pixelheight'= 100),
    RadioButton('fillcolor'="#ff0000"),
    Slider('fillcolor'="#ff0000"),
    TextArea("I love it",'fillcolor'="#ff0000")
  ))
```

```
)  
)  
):
```



▼ Example: Interactive Table

Consider the following example that uses coloring to delimit the rows. The contents inside of the Table are live active components. The numbers in the 'n' column are housed in [TextArea](#) components, and the series approximations are housed in [MathContainer](#) components. All components have their background color set to match the Table cells' fill color.

This example is interactive. Try changing $n=6$ to $n=8$ by putting your cursor on the 6 and typing over it with 8.

```
TaylorTable( exp(x), x, 6 );
```

n	taylor(exp(x),x,n)
1	$1 + O(x)$
2	$1 + x + O(x^2)$
3	$1 + x + \frac{1}{2} x^2 + O(x^3)$
4	$1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + O(x^4)$
5	$1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{1}{24} x^4 + O(x^5)$
6	$1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{1}{24} x^4 + \frac{1}{120} x^5 + O(x^6)$

The following Code Edit Region implements the `TaylorTable` command seen above:

```

1 TaylorTable := proc( fn, var, ord )
2     uses DocumentTools, DocumentTools:-Layout, DocumentTools:-Components;
3     local Colors := Array(0..1,[white,"#ccddff"]);
4     InsertContent(
5         Worksheet(
6             Table( 'interior'='none', 'width'=600, 'widthmode'=pixels,
7                 Column( 'weight'=10), Column( 'weight'=90),
8                 Row(
9                     Cell( Textfield(Font("n",color=white,bold)), fillcolor=blue ),
10                    Cell( Textfield(Font(sprintf("taylor(%a,%a,n)",fn,var),color=white,bold)), fillcolor=blue )
11                ),
12                seq(
13                    Row(
14                        Cell(`fillcolor`=Colors[row mod 2],
15                            TextArea(convert(row,string),
16                                'identity'=cat("TA",row),
17                                'action'=sprintf("DocumentTools:-Do( %%MC%d = taylor(%a,%a,%%TA%d) ),'how,fn,var,row),
18                                'showborders'=false,
19                                'fillcolor`=Colors[row mod 2] )
20                    ),
21                    Cell(`fillcolor`=Colors[row mod 2],
22                        MathContainer(taylor(fn,var,row),
23                            'identity'=cat("MC",row),
24                            'minwidth'=`if`(col=1,60,540),
25                            'minheight'=40,
26                            'autofit',
27                            'showborders'=false,
28                            'fillcolor`=Colors[row mod 2] )
29                )
30            ),
31            row = 1..ord )
32        )
33    );
34    NULL;
35 end proc;

```

▼ **Programmatic Content Generation**

There have also been several new additions that build on the framework for programmatic content generation, including the ability to replace existing worksheet content and add a notion of state to components.

For more details on Component State, see the what's new page for [Component State](#).
For more details on Replaceable Content, see the what's new page for [Replaceable Content](#).