Tables

Introduction

Tables are objects that can be used to position information in specific and fixed locations in a script's visual space. Contrary to all other plots or objects drawn in Pine Script™, tables are not anchored to specific bars; they *float* in a script's space, whether in overlay or pane mode, in studies or strategies, independently of the chart bars being viewed or the zoom factor used.

Tables contain cells arranged in columns and rows, much like a spreadsheet. They are created and populated in two distincts steps:

1. A table's structure and key attributes are defined using table.new(), which returns a table ID that acts like a pointer to the table, just like label, line, or array IDs do. The table.new() call will create the table object but does not display it.
2. Once created, and for it to display, the table must be populated using one table.cell() call for each cell. Table cells can contain text, or not. This second step is when the width and height of cells are defined.

Most attributes of a previously created table can be changed using `table.set_*()` setter functions. Attributes of previously populated cells can be modified using `table.cell_set_*()` functions.

A table is positioned in an indicator's space by anchoring it to one of nine references: the four corners or midpoints, including the center. Tables are positioned by expanding the table from its anchor, so a table anchored to the position.middle_right reference will be drawn by expanding up, down and left from that anchor.

Two modes are available to determine the width/height of table cells:

- A default automatic mode calculates the width/height of cells in a column/row using the widest/highest text in them.
- An explicit mode allows programmers to define the width/height of cells using a percentage of the indicator's available x/y space.

Displayed table contents always represent the last state of the table, as it was drawn on the script's last execution, on the dataset's last bar. Contrary to values displayed in the Data Window or in indicator values, variable contents displayed in tables will thus not change as a script user moves his cursor over specific chart bars. For this reason, it is

strongly recommended to always restrict execution of all `table.*()` calls to either the first or last bars of the dataset. Accordingly:

- Use the var keyword to declare tables.
- Enclose all other calls inside an if barstate.islast block.

Multiple tables can be used in one script, as long as they are each anchored to a different position. Each table object is identified by its own ID. Limits on the quantity of cells in all tables are determined by the total number of cells used in one script.

Creating tables

When creating a table using table.new(), three parameters are mandatory: the table's position and its number of columns and rows. Five other parameters are optional: the table's background color, the color and width of the table's outer frame, and the color and width of the borders around all cells, excluding the outer frame. All table attributes except its number of columns and rows can be modified using setter functions: table.set_position(), table.set_bgcolor(), table.set_frame_color(), table.set_frame_width(), table.set_border_color() and table.set_border_width().

Tables can be deleted using table.delete(), and their content can be selectively removed using table.clear().

When populating cells using table.cell(), you must supply an argument for four mandatory parameters: the table id the cell belongs to, its column and row index using indices that start at zero, and the text string the cell contains, which can be null. Seven other parameters are optional: the width and height of the cell, the text's attributes (color, horizontal and vertical alignment, size), and the cell's background color. All cell attributes can be modified using setter functions: table.cell_set_text(), table.cell_set_width(), table.cell_set_height(), table.cell_set_text_color(), table.cell_set_text_halign(), table.cell_set_text_valign(), table.cell_set_text_size() and table.cell_set_bgcolor().

Keep in mind that each successive call to table.cell() redefines **all** the cell's properties, deleting any properties set by previous table.cell() calls on the same cell.

Placing a single value in a fixed position

Let's create our first table, which will place the value of ATR in the upper-right corner of the chart. We first create a one-cell table, then populate that cell:

```
//@version=5
indicator("ATR", "", true)
// We use `var` to only initialize the table on the first bar.
var table atrDisplay = table.new(position.top_right, 1, 1)
// We call `ta.atr()` outside the `if` block so it executes on each bar.
myAtr = ta.atr(14)
if barstate.islast
    // We only populate the table on the last bar.
    table.cell(atrDisplay, 0, 0, str.tostring(myAtr))
```

Note that:

- We use the var keyword when creating the table with table.new().
- We populate the cell inside an if barstate.islast block using table.cell().
- When populating the cell, we do not specify the `width` or `height`. The width and height of our cell will thus adjust automatically to the text it contains.
- We call `ta.atr(14)` prior to entry in our if block so that it evaluates on each bar. Had we used `str.tostring(ta.atr(14))` inside the if block, the function would not have evaluated correctly because it would be called on the dataset's last bar without having calculated the necessary values from the previous bars.

Let's improve the usability and aesthethics of our script:

```
//@version=5
indicator("ATR", "", true)
atrPeriodInput = input.int(14,  "ATR period", minval = 1, tooltip = "Using a period of

var table atrDisplay = table.new(position.top_right, 1, 1, bgcolor = color.gray, frame_
myAtr = ta.atr(atrPeriodInput)
if barstate.islast
    table.cell(atrDisplay, 0, 0, str.tostring(myAtr, format.mintick), text_color = colo
```

Note that:

- We used table.new() to define a background color, a frame color and its width.
- When populating the cell with table.cell(), we set the text to display in white.
- We pass format.mintick as a second argument to the str.tostring() function to restrict the precision of ATR to the chart's tick precision.
- We now use an input to allow the script user to specify the period of ATR. The input also includes a tooltip, which the user can see when he hovers over the "i" icon in the script's "Settings/Inputs" tab.

Coloring the chart's background

This example uses a one-cell table to color the chart's background on the bull/bear state of RSI:

```
//@version=5
indicator("Chart background", "", true)
bullColorInput = input.color(color.new(color.green, 95), "Bull", inline = "1")
bearColorInput = input.color(color.new(color.red, 95), "Bear", inline = "1")
// ———— Function colors chart bg on RSI bull/bear state.
colorChartBg(bullColor, bearColor) =>
    var table bgTable = table.new(position.middle_center, 1, 1)
    float r = ta.rsi(close, 20)
    color bgColor = r > 50 ? bullColor : r < 50 ? bearColor : na
    if barstate.islast
        table.cell(bgTable, 0, 0, width = 100, height = 100, bgcolor = bgColor)

colorChartBg(bullColorInput, bearColorInput)
```

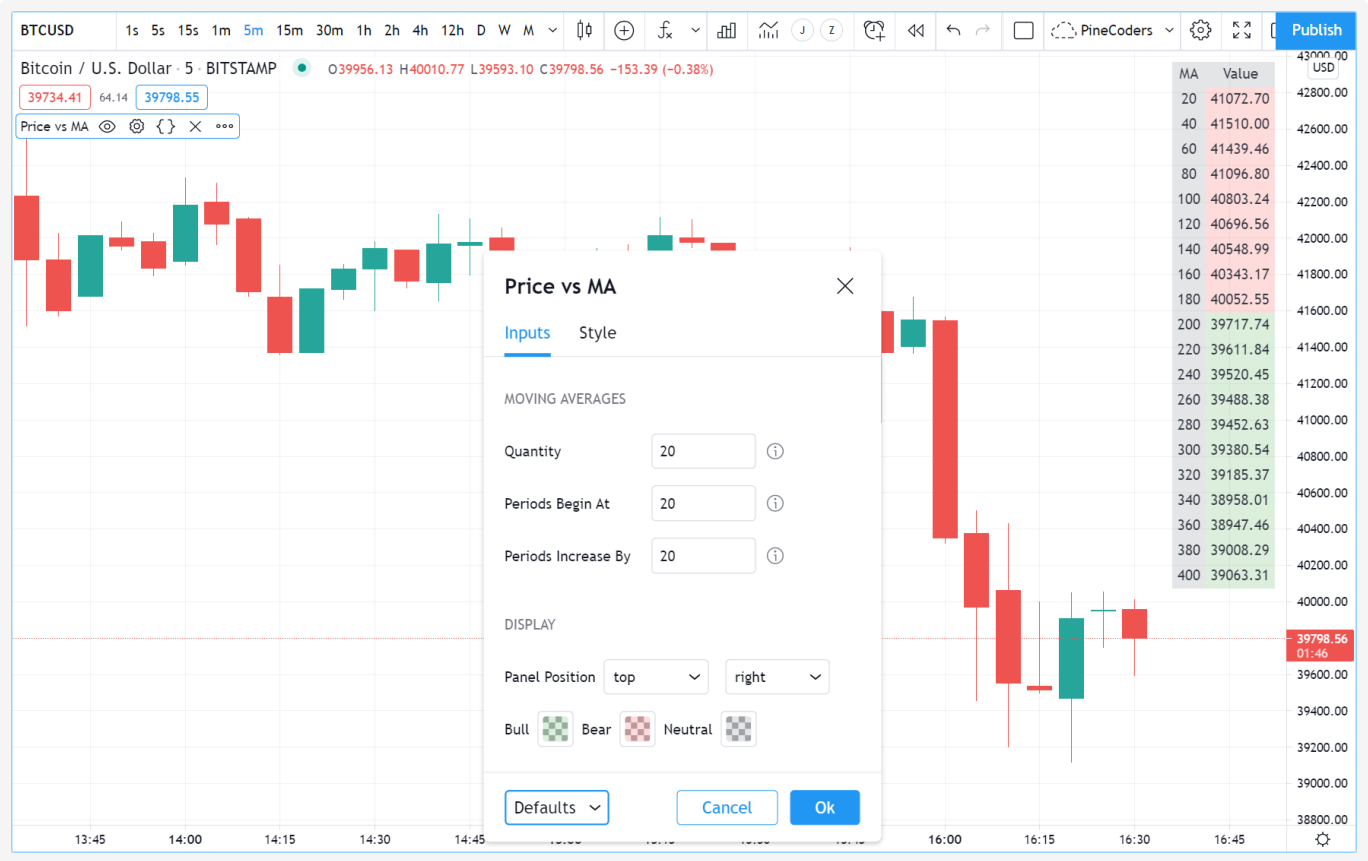
Note that:

- We provide users with inputs allowing them to specify the bull/bear colors to use for the background, and send those input colors as arguments to our `f_colorChartBg()` function.
- We create a new table only once, using the var keyword to declare the table.
- We use table.cell() on the last bar only, to specify the cell's properties. We make the cell the width and height of the indicator's space, so it covers the whole chart.

Creating a display panel

Tables are ideal to create sophisticated display panels. Not only do they make it possible for display panels to always be visible in a constant position, they provide more flexible formatting because each cell's properties are controlled separately: background, text color, size and alignment, etc.

Here, we create a basic display panel showing a user-selected quantity of MAs values. We display their period in the first column, then their value with a green/red/gray background that varies with price's position with regards to each MA. When price is above/below the MA, the cell's background is colored with the bull/bear color. When the MA falls between the current bar's open and close, the cell's background is of the neutral color.



```
//@version=5
indicator("Price vs MA", "", true)

var string GP1 = "Moving averages"
int     masQtyInput    = input.int(20, "Quantity", minval = 1, maxval = 40, group = GP
int     masStartInput  = input.int(20, "Periods begin at", minval = 2, maxval = 200, g
int     masStepInput   = input.int(20, "Periods increase by", minval = 1, maxval = 100,

var string GP2 = "Display"
string  tableYposInput = input.string("top", "Panel position", inline = "11", options =
string  tableXposInput = input.string("right", "", inline = "11", options = ["left", "
color   bullColorInput = input.color(color.new(color.green, 30), "Bull", inline = "12",
color   bearColorInput = input.color(color.new(color.red, 30), "Bear", inline = "12", g
color   neutColorInput = input.color(color.new(color.gray, 30), "Neutral", inline = "12

var table panel = table.new(tableYposInput + "_" + tableXposInput, 2, masQtyInput + 1)
if barstate.islast
    // Table header.
    table.cell(panel, 0, 0, "MA", bgcolor = neutColorInput)
    table.cell(panel, 1, 0, "Value", bgcolor = neutColorInput)

int period = masStartInput
for i = 1 to masQtyInput
    // ————— Call MAs on each bar.
    float ma = ta.sma(close, period)
    // ————— Only execute table code on last bar.
    if barstate.islast
        // Period in left column.
        table.cell(panel, 0, i, str.tostring(period), bgcolor = neutColorInput)
        // If MA is between the open and close, use neutral color. If close is lower/hi
        bgColor = close > ma ? open < ma ? neutColorInput : bullColorInput : open > ma
        // MA value in right column.
        table.cell(panel, 1, i, str.tostring(ma, format.mintick), text_color = color.bl
    period += masStepInput
```

Note that:

- Users can select the table's position from the inputs, as well as the bull/bear/neutral colors to be used for the background of the right column's cells.
- The table's quantity of rows is determined using the number of MAs the user chooses to display. We add one row for the column headers.
- Even though we populate the table cells on the last bar only, we need to execute the calls to ta.sma() on every bar so they produce the correct results. The compiler warning that appears when you compile the code can be safely ignored.
- We separate our inputs in two sections using `group`, and join the relevant ones on the same line using `inline`. We supply tooltips to document the limits of certain fields using `tooltip`.

Displaying a heatmap

Our next project is a heatmap, which will indicate the bull/bear relationship of the current price relative to its past values. To do so, we will use a table positioned at the bottom of the chart. We will display colors only, so our table will contain no text; we will simply color the background of its cells to produce our heatmap. The heatmap uses a user-selectable lookback period. It loops across that period to determine if price is above/below each bar in that past, and displays a progressively lighter intensity of the bull/bear color as we go further in the past:


```
//@version=5
indicator("Price vs Past", "", true)

var int MAX_LOOKBACK = 300

int     lookBackInput  = input.int(150, minval = 1, maxval = MAX_LOOKBACK, step = 10)
color   bullColorInput = input.color(#00FF00ff, "Bull", inline = "11")
color   bearColorInput = input.color(#FF0080ff, "Bear", inline = "11")

// ———— Function draws a heatmap showing the position of the current `_src` relative t
drawHeatmap(src, lookBack) =>
    // float src      : evaluated price series.
    // int   lookBack: number of past bars evaluated.
    // Dependency: MAX_LOOKBACK

    // Force historical buffer to a sufficient size.
    max_bars_back(src, MAX_LOOKBACK)
    // Only run table code on last bar.
    if barstate.islast
        var heatmap = table.new(position.bottom_center, lookBack, 1)
        for i = 1 to lookBackInput
            float transp = 100. * i / lookBack
            if src > src[i]
                table.cell(heatmap, lookBack - i, 0, bgcolor = color.new(bullColorInput
            else
                table.cell(heatmap, lookBack - i, 0, bgcolor = color.new(bearColorInput

drawHeatmap(high, lookBackInput)
```

Note that:

- We define a maximum lookback period as a `MAX_LOOKBACK` constant. This is an important value and we use it for two purposes: to specify the number of columns we will create in our one-row table, and to specify the lookback period required for the `_src` argument in our function, so that we force Pine Script™ to create a historical buffer size that will allow us to refer to the required quantity of past values of `_src` in our for loop.
- We offer users the possibility of configuring the bull/bear colors in the inputs and we use `inline` to place the color selections on the same line.
- Inside our function, we enclose our table-creation code in an if barstate.islast construct so that it only runs on the last bar of the chart.
- The initialization of the table is done inside the if statement. Because of that, and the fact that it uses the var keyword, initialization only occurs the first time the script executes on a last bar. Note that this behavior is different from the usual var declarations in the script's global scope, where initialization occurs on the first bar of the dataset, at bar_index zero.
- We do not specify an argument to the `text` parameter in our table.cell() calls, so an empty string is used.

- We calculate our transparency in such a way that the intensity of the colors decreases as we go further in history.
- We use dynamic color generation to create different transparencies of our base colors as needed.
- Contrary to other objects displayed in Pine scripts, this heatmap's cells are not linked to chart bars. The configured lookback period determines how many table cells the heatmap contains, and the heatmap will not change as the chart is panned horizontally, or scaled.
- The maximum number of cells that can be displayed in the scritp's visual space will depend on your viewing device's resolution and the portion of the display used by your chart. Higher resolution screens and wider windows will allow more table cells to be displayed.

Tips

- When creating tables in strategy scripts, keep in mind that unless the strategy uses `calc_on_every_tick = true`, table code enclosed in if barstate.islast blocks will not execute on each realtime update, so the table will not display as you expect.
- Keep in mind that successive calls to table.cell() overwrite the cell's properties specified by previous table.cell() calls. Use the setter functions to modify a cell's properties.
- Remember to control the execution of your table code wisely by restricting it to the necessary bars only. This saves server resources and your charts will display faster, so everybody wins.

Strategies

Text and shapes