Basics about column sizing and cell contents

This is GRID_STYLE with explicit column widths. Each cell contains a string or number

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mugs	0	4	17	3	21	47	12	33	2	-2	44	89
T-Shirts	0	42	9	-3	16	4	72	89	3	19	32	119
Miscellaneous accessories	0	0	0	0	0	0	1	0	0	0	2	13
Hats	893	912	1,212	643	789	159	888	1,298	832	453	1,344	2,843

This is GRID_STYLE with no size info. It does the sizes itself, measuring each text string and computing the space it needs. If the text is too wide for the frame, the table will overflow as seen here.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mugs	0	4	17	3	21	47	12	33	2	-2	44	89
T-Shirts	0	42	9	-3	16	4	72	89	3	19	32	119
Miscellaneous accessories	0	0	0	0	0	0	1	0	0	0	2	13
Hats	893	912	1,212	643	789	159	888	1,298	832	453	1,344	2,843

This demonstrates the effect of adding text strings with newlines to a cell. It breaks where you specify, and if rowHeights is None (i.e automatic) then you'll see the effect. See bottom left cell.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mugs	0	4	17	3	21	47	12	33	2	-2	44	89
T-Shirts	0	42	9	-3	16	4	72	89	3	19	32	119
Key Ring	0	0	0	0	0	0	1	0	0	0	2	13
Hats Large	893	912	1,212	643	789	159	888	1,298	832	453	1,344	2,843

This table does not specify the size of the first column, so should work out a sane one. In this case the element at bottom left is a paragraph, which has no intrinsic size (the height and width are a function of each other). So, it tots up the extra space in the frame and divides it between any such unsizeable columns. As a result the table fills the width of the frame (except for the 6 point padding on either size).

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mugs	0	4	17	3	21	47	12	33	2	-2	44	89
T-Shirts	0	42	9	-3	16	4	72	89	3	19	32	119
Key Ring	0	0	0	0	0	0	1	0	0	0	2	13
Let's **really mess things up with a *paragraph***	893	912	1,212	643	789	159	888	1,298	832	453	1,344	2,843

Row and Column spanning

This shows a very basic table. We do a faint pink grid to show what's behind it - imagine this is not printed, as we'll overlay it later with some black lines. We're going to "span" some cells, and have put a value of None in the data to signify the cells we don't care about. (In real life if you want an empty cell, put '' in it rather than None).

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
North	Eggs	101	111	121	131	464
North	Guinness	102	112	122	132	468
South	Spam	100	110	120	130	460
South	Eggs	101	111	121	131	464
South	Guinness	102	112	122	132	468

We now center the text for the "period" across the four cells for each quarter. To do this we add a 'span' command to the style to make the cell at row 1 column 3 cover 4 cells, and a 'center' command for all cells in the top row. The spanning is not immediately evident but trust us, it's happening - the word 'Period' is centered across the 4 columns. Note also that the underlying grid shows through. All line drawing commands apply to the underlying grid, so you have to take care what you put grids through.

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
North	Eggs	101	111	121	131	464
North	Guinness	102	112	122	132	468
South	Spam	100	110	120	130	460
South	Eggs	101	111	121	131	464
South	Guinness	102	112	122	132	468

We repeat this for the words 'Region', Product' and 'Total', which each span the top 2 rows; and for 'Nprth' and 'South' which span 3 rows. At the moment each cell's alignment is the default (bottom), so these words appear to have "dropped down"; in fact they are sitting on the bottom of their allocated ranges. You will just see that all the 'None' values vanished, as those cells are not drawn any more.

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468
South	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

Now we'll tart things up a bit. First, we set the vertical alignment of each spanned cell to 'middle'. Next we add in some line drawing commands which do not slash across the spanned cells (this needs a bit of work). Finally we'll add some thicker lines to divide it up, and hide the pink. Voila!

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468
South	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

How cells get sized

So far the table has been auto-sized. This can be computationally expensive, and can lead to yucky effects. Imagine a lot of numbers, one of which goes to 4 figures - tha numeric column will be wider. The best approach is to specify the column widths where you know them, and let the system do the heights. Here we set some widths - an inch for the text columns and half an inch for the numeric ones.

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468
South	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

The auto-sized example 2 steps back demonstrates one advanced feature of the sizing algorithm. In the table below, the columns for Q1-Q4 should all be the same width. We've made the text above it a bit longer than "Period". Note that this text is technically in the 3rd column; on our first implementation this was sized and column 3 was therefore quite wide. To get it right, we ensure that any cells which span columns, or which are 'overwritten' by cells which span columns, are assigned zero width in the cell sizing. Thus, only the string 'Q1' and the numbers below it are calculated in estimating the width of column 3, and the phrase "What time of year?" is not used. However, row-spanned cells are taken into account. ALL the cells in the leftmost column have a vertical span (or are occluded by others which do) but it can still work out a sane width for them.

Region	Product	Which time of year?				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

South	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

Paragraphs and unsizeable objects in table cells.

Paragraphs and other flowable objects make table sizing much harder. In general the height of a paragraph is a function of its width so you can't ask it how wide it wants to be - and the REALLY wide all-on-one-line solution is rarely what is wanted. We refer to Paragraphs and their kin as "unsizeable objects". In this example we have set the widths of all but the first column. As you can see it uses all the available space across the page for the first column. Note also that this fairly large cell does NOT contribute to the height calculation for its 'row'. Under the hood it is in the same row as the second Spam, but this row gets a height based on its own contents and not the cell with the paragraph.

Region	Product	Period				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468
Let's **really mess things up** with a *paragraph*, whose height is a function of the width you give it.	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

This one demonstrates that our current algorithm does not cover all cases :-(The height of row 0 is being driven by the width of the para, which thinks it should fit in 1 column and not 4. To really get this right would involve multiple passes through all the cells applying rules until everything which can be sized is sized (possibly backtracking), applying increasingly dumb and brutal rules on each pass.

Region	Product	Let's **really mess things up** with a *paragraph*.				Total
		Q1	Q2	Q3	Q4	
North	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468
Let's **really mess things up** with a *paragraph*, whose height is a function of the width you give it.	Spam	100	110	120	130	460
	Eggs	101	111	121	131	464
	Guinness	102	112	122	132	468

To avoid these problems remember the golden rule of ReportLab tables: (1) fix the widths if you can, (2) don't use a paragraph when a string will do.

Unsized columns that contain flowables without precise widths, such as paragraphs and nested tables, still need to try and keep their content within borders and ideally even honor percentage requests. This can be tricky--and expensive. But sometimes you can't follow the golden rules.

The code first calculates the minimum width for each unsized column by iterating over every flowable in each column and remembering the largest minimum width. It then allocates available space to accomodate the minimum widths. Any remaining space is divided up, treating a width of '*' as greedy, a width of None as non-greedy, and a percentage as a weight. If a column is already

wider than its percentage warrants, it is not further expanded, and the other widths accomodate it.

For instance, consider this tortured table. It contains four columns, with widths of None, None, 60%, and 20%, respectively, and a single row. The first cell contains a paragraph. The second cell contains a table with fixed column widths that total about 50% of the total available table width. The third cell contains a string. The last cell contains a table with no set widths but a single cell containing a paragraph.

This is a paragraph. The column has a width of None.	This table is set to take up two and a half inches. The column that holds it has a width of None.	60% width	This is a table with a paragraph in it but no width set. The column width in the containing table is 20%.

Notice that the second column does expand to account for the minimum size of its contents; and that the remaining space goes to the third column, in an attempt to honor the '60%' request as much as possible. This is reminiscent of the typical HTML browser approach to tables.

To get an idea of how potentially expensive this is, consider the case of the last column: the table gets the minimum width of every flowable of every cell in the column. In this case one of the flowables is a table with a column without a set width, so the nested table must itself iterate over its flowables. The contained paragraph then calculates the width of every word in it to see what the biggest word is, given the set font face and size. It is easy to imagine creating a structure of this sort that took an unacceptably large amount of time to calculate. Remember the golden rule, if you can.

This code does not yet handle spans well.