Calc: Load data with macros

Tried of trawling through stock exchange data only to find that you definitely won't be buying that yacht today? Macros will make your life much easier...

Hopefully your stocks and shares are doing better than mine. If not, you're probably sitting sobbing into your beer, but don't be too downhearted. Here at *Linux Format* we may not be able to improve your market-forecasting skills, but at least we can help you see how badly you're doing more easily.

This is where *OpenOffice.org* comes in. In this tutorial, we're going to look at how to download financial data from a website directly into its spreadsheet component *Calc*, then we'll see how to turn that data into a chart, and to top it all, we'll explore how to do all of that automatically. In fact, you'll learn how to do all of this without even having to open up *OOo* (apart from viewing the end result, of course). Well, when I say you won't have to open up *OOo*, you will have to write a macro or two – but then you'll be able to process the data without having to open up *OOo*.

Anyway, our first step is going to be to find the online financial data. Now, you may have your own sources – but in this case we're going to be using the financial portal Yahoo Finance. If you don't happen to have an interest in the stock market, don't turn the page: the skills you'll learn don't have to be used with stocks and shares, so you can go on to use them with many other data sources. For the purposes of this tutorial, however, greed is good!

Obtaining the data

You'll find Yahoo Finance at **http://finance.yahoo.com**, and you'll also find that it's very easy to use – there's a box containing the words Enter Symbol(s) and a button named Get Quotes. All you have to do is to enter the symbol for the company you're interested in, and Yahoo Finance will provide a link (Symbol Lookup) that will give you the symbol for the company. For example, the symbol for Microsoft on the London Stock Exchange is MSF.L. Once you've entered this symbol and pressed Get Quotes, you'll see the most current results for the company. And now we getting to the bit that we're really interested in – the download.

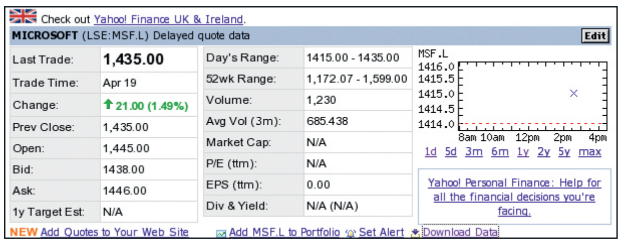
If you look at the quotes screen, you'll see a link: Download Data. From this we can obtain the URL for the data itself: **http://download.finance.yahoo.com/d/quotes.csv?s=MSF.L&f=sl1d1t1c1ohgv&e=.csv**. And why do we want the URL? Because we can now type this directly into the web browser at any time. It's not a major time saver – but it's a start.

It's this next step we can make the big saving on. If you click on Download Data or type in the URL, *Calc* will load the CSV file into its Import wizard. With this you tell *OOo* about the separators that the file uses (obviously commas in this case), and then *OOo* will place each of the fields in its own column. So, nothing new so far – you've probably done this type of thing hundreds of times before. However, we can improve this process by removing our manual input, and to do that we're going to have to write a macro.

The first macro

You've already learned that we can obtain the stock market quote by entering a symbol representing a company as part of a URL into your web browser. You can then load this as a CSV file into *OOo Calc* via its Import wizard. Now, before we write our macro to carry out our import, let's just examine that data we're actually going to be importing. You'll find that the fields that are downloads are defined by the **f** parameter in the URL. If you experiment with them you'll be able to work out what each one does, but we're only interested in three of then: **s**, the company name, **l1**, the last trading price, and **d1**, the last trade date. Our URL, therefore, becomes **http://download.finance.yahoo.com/d/quotes.csv?s=MSF.L&f=sl1d1&e=.csv**, and that's what we'll use in the macro.

OK – we're down to writing the macro. Open *OOo* (it doesn't matter which application – *Calc* or *Writer* will be fine), then click on Tool > Macros > Organize Macros > OpenOffice.org Basic. You'll see that you have a selection of languages to work in (Basic, Python, BeanShell or JavaScript), but it's Basic that we're interested in. A dialog box will open, and at this point you'll need to press the Organizer button. Another dialog will open, and you'll see that you're in the Module tab with Standard selected. At this point you may be wondering what on earth is going on – but it's


Check out Yahoo! Finance UK & Ireland.

MICROSOFT (LSE:MSF.L) Delayed quote data

Last Trade:	1,435.00	Day's Range:	1415.00 - 1435.00
Trade Time:	Apr 19	52wk Range:	1,172.07 - 1,599.00
Change:	↑ 21.00 (1.49%)	Volume:	1,230
Prev Close:	1,435.00	Avg Vol (3m):	685.438
Open:	1,445.00	Market Cap:	N/A
Bid:	1438.00	P/E (ttm):	N/A
Ask:	1446.00	EPS (ttm):	0.00
1y Target Est:	N/A	Div & Yield:	N/A (N/A)

Yahoo! Personal Finance: Help for all the financial decisions you're facing.

NEW Add Quotes to Your Web Site Add MSF.L to Portfolio Set Alert Download Data

› **You can download stock exchange data as a CSV file from Yahoo Finance into *OpenOffice.org*'s spreadsheet component.**

Using crontab

If you're new to **crontab**, the fields and stars may look a little confusing. However, the field order is quite simple really:

1 **Minute** (0-59)
2 **Hour** (0-23)
3 **Day of the month** (1-31)
4 **Month** (1-12)
5 **Day of the week** (0-6 with 0 as representing Sunday)
6 **The command to be run**

The star means "run for every occasion" – so * in the third field means "run on every day of the month". You can also use combinations of numbers in the fields, so (again in the 3rd field) 1,3,10-20 would mean "run on the 1st, the 3rd and every day from the 10th to the 20th".

quite simple, really. *Standard* is a library, and a library is used to store groups of modules. Modules are used to store groups of macros. If you've got that, click on New to create your own, brand-new module in the *Standard* library.

When you hit New you'll see that *OOo* suggests a name for your module: **Module1**. Don't use this. It gets very tedious having modules named Module1, Module2, Module3, Module4 *ad infinitum*. Instead give it a meaningful name – for instance, I named the module **lxf94**. With this new module in place click on Edit, and *OOo* will take you to the Basic edit screen – and it's here that we'll create the macro itself.

Actually, you'll find that *OOo* will have already created a macro for you, a subroutine named **Main**. Before we go any further, don't confuse this with the Main that you see in other programming languages: it's simply a blank macro so that *OOo* has something to work with if you happen to press the run button (the button showing the corner of a page and a downwards pointing arrow). In fact, *OOo* will run the first macro that it finds in the module. So, for example, you could place **Main1** above **Main** and that would be run instead.

With that said we're ready to write the macro itself (place this after the **Main** subroutine):

```
Function open_csv_file (url as String) as Object
Dim oProperty(0) as New com.sun.star.beans.PropertyValue
oProperty(0).Name = "FilterOptions"
oProperty(0).Value = "44"
open_csv_file = starDeskTop.loadComponentFromUrl(url, "_
blank", 0, oProperty())
End Sub
```

Here we've created a function that will load any CSV file without the need to use the import wizard. It does this by making use of the **FilterOptions** property, and setting it to 44. Why 44? Because 44 is the ASCII code for – yes, you've guessed it – a comma. And why are we using a function here? It's because **open_csv_file** creates an object – the document itself – and we'll need to access this object from our code.

Now, if you try running your code, nothing will happen. That's simply because **Main** doesn't do anything yet, so let's get it to do something:

```
Sub Main
Dim oDoc as Object
oDoc = open_csv_file _
           ("http://download.finance.yahoo.com/d/quotes.
csv?s=MSF.L&f=sd1l1&e=.csv")
End Sub
```

And now, when you click the run button something will happen: after a moment or two (depending on your connection speed) a *Calc* file will appear containing the last trading price for Microsoft on the London Stock Exchange. That's nice, but you'll be thinking, "I don't want to have to do this for each of my companies – I want to run it once for my whole portfolio." Fine, that's what we'll look at next.

A macro for multiple inputs

We've seen how easy it is to automate the loading of the data for a single company. In order to do the same for a number of companies we just need to obtain the new symbol and add it to the URL. For example, if we're interested in Novell we need to add the symbol **NOVL**, and our URL becomes **http://download.finance.yahoo.com/d/quotes.csv?s=MSF.L&s=NOVL&f=sd1l1&e=.csv**. We just have to modify our code to create this new information. First we'll add a new subroutine to keep our lives nice and simple:

```
Sub download_stock_price(companySymbols)
Dim oDoc as Object
Dim cSymbols as String, oUrl as String
cSymbols = join(companySymbols, "&s=")
oUrl = "http://download.finance.yahoo.com/d/quotes.csv?s=" _
     & cSymbols & "&f=sl1d1&e=.csv"
oDoc = open_csv_file(oUrl)
End Sub
```

You'll notice that this new macro takes **companySymbols** as an input – this is actually going to be an array containing the list of company symbols.

The subroutine joins all of the symbols as a single string (with **&s=** inserted between each symbol), and then creates the correct URL to be used with the **open_csv_file** subroutine that we created earlier. Having created this new macro, we need to change the **Main** subroutine so that we can send the array of company symbols to it:

```
Sub Main
download_stock_price(array("MSF.L","NOVL"))
End Sub
```

When you hit the run button this time, you'll end up with a *Calc* file containing details for all of the companies that you've listed in the array – and, of course, you can add as few or as many company symbols as you want.

Saving the file

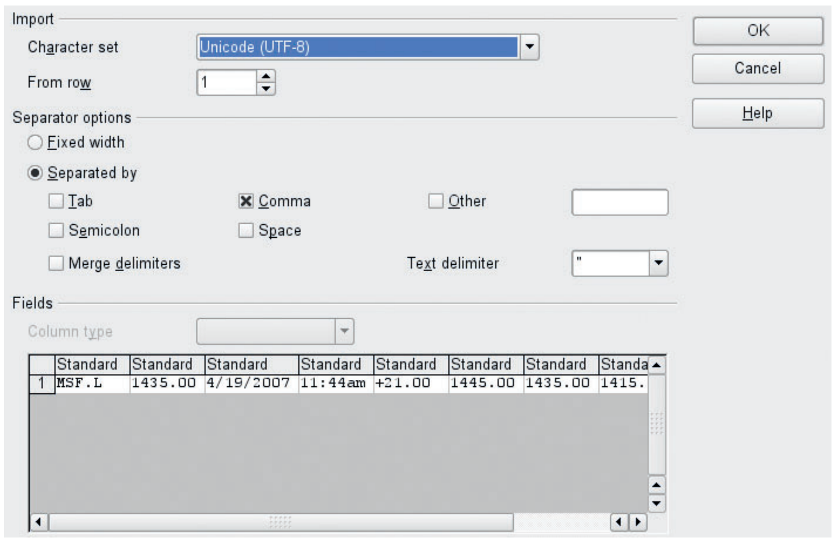
The next stage of our automation is to save the file to the hard drive. To do this we need another simple subroutine:

```
Sub save_file_as_ods(doc as Object, directory as String, prefix as
String)
Dim oUrl as String
oUrl  = convertToUrl(directory & "/" & prefix & ".ods")
doc.storeAsUrl(oUrl,Array())
```

»

Quick tip

In this tutorial, we've made use of the daily snapshots, but you may want to investigate the historical data for each company.



Character set: Unicode (UTF-8)
From row: 1
( ) Fixed width
(●) Separated by
[ ] Tab [✓ Comma] [ ] Other
[ ] Semicolon [ ] Space
[ ] Merge delimiters
Text delimiter: "
Column type:

	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standa
1	MSF.L	1435.00	4/19/2007	11:44am	+21.00	1445.00	1435.00	1415.

› If you don't use a macro, you'll have to go through *Calc*'s Import wizard. A lot!

```
End Sub
```

There's a couple of things to take notice of here: firstly, the **convertToUrl** function. When opening and closing files, the macro needs the file name to be in a particular format – and **convertToUrl** does this conversion for you. Secondly, you'll see that as well as the directory and file prefix (that is, the filename without **.ods**) to the subroutine we also pass **doc**. This is the name of the object that we create with the **open_csv_file** function.

Next, you need to make a little change to **download_stock_price**. Add the following line to the end of the subroutine:

```
save_file_as_ods(oDoc, "/tmp", "test_lxf94")
```

This time, when you run **Main** you'll find that your document will be named 'test_lxf94', and if you look in the **/tmp** directory you'll find a file entitled **test_lxf94.ods**, which (if you open it) will contain your newly downloaded data.

Quick tip

A quick way to get to the Organizer is to click on Tools > Macros > Organize Dialogs, and then go to the Module tab.

To the command line!

At the start of this article I said that you'd be able to do all of this without opening *OOo* (apart from viewing the end result), so let's look at how we can achieve that. Having seen how to automatically open and save our file, you'll now see how to do that without any visual output. We'll start by adding a line of code to the end of the **download_stock_price** macro:

```
oDoc.close(true)
```

All that does is close your document. If you do run **Main** at the moment you'll just see the document open and immediately close. Why we have done this first? Simply to ensure that we don't end up with any session left hidden in the background when we make the next change, which will be to hide our document.

You've already learned that we can open a CSV file without using the import wizard by setting the appropriate properties, and so I expect you'll realise that opening a document in Hidden mode is just the same. Therefore, it's back to the **open_csv_file** function. First we need to redefine the number of properties that we're going to be using:

```
Dim oProperty(1) as New com.sun.star.beans.PropertyValue
```

Then we need to add the details for the new property:

```
oProperty(1).Name = "Hidden"
oProperty(1).Value = True
```

Optional inputs for macros

In the tutorial, we switched off the Hidden mode, but if you're going to be doing this a lot you may want to change the code to make this optional:

```
Function open_csv_file (url as String, Optional show_form as Boolean) as Object
 Dim oProperty(1) as New com.sun.star.beans.PropertyValue
 if ismissing(show_form) then
  show_form = False
 end if
 oProperty(0).Name = "FilterOptions"
 oProperty(0).Value = "44"
 oProperty(1).Name = "Hidden"
 if (show_form) then
   oProperty(1).Value = False
 else
   oProperty(1).Value = True
 end if
 open_csv_file = starDeskTop.loadComponentFromUrl(url, "_blank", 0, oProperty())
 End Function
```

Now remember that, by default, the document will remain invisible, but if you want to see it, you can simply change the code in **open_csv_file** to

```
oDoc = open_csv_file(oUrl, True)
```

Once you've made the change to the code, press the run button... and nothing happens. Worry not. Check the file details:

```
bainm@aeneas:~> ls -l /tmp/test_lxf94.ods
-rw-r--r-- 1 bainm users 6455 2007-04-23 09:00 /tmp/test_lxf94.ods
```

Now run the macro and check again. You'll see that the file does actually update:

```
bainm@aeneas:~> ls -l /tmp/test_lxf94.ods
-rw-r--r-- 1 bainm users 6454 2007-04-23 09:01 /tmp/test_lxf94.ods
```

This means that we can run the process invisibly, but (at the moment) we still need to have *OOo* open in order to do this. Well, no, we don't. Close all instances of *OOo*, and then type the following on the command line:

```
scalc -headless "macro:///Standard.lxf94.Main"
```

Absolutely nothing will happen – you won't even see the *OOo* introduction screen. However, if you do an **ls -l** on the file again you'll see that the file has been updated.

Now we can complete the automation task by creating a *Cron* job. Once we've done that you won't have to worry about running the macro. For example, you may want to set the *Cron* job to run every day at 8:45am so that the new file is ready for you to view as soon as you get in the office at 9:00 am. To do this, go to the command line. Type **crontab -e**, and then

```
45 8 * * * scalc -headless "macro:///Standard.lxf94.Main"
```

Next, press Ctrl-D to save the *Cron* file. And that's all there is to it – your stock market download is now totally automated.

Introducing charts

Everyone knows the saying "a picture paints a thousand words", and that's true of charts as well. I don't know about you, but I find it much easier to understand a graph than just a mass of figures – and that's all we've got at the moment. So let's transform our data.

Now you do know how to create a chart in *OOo*, don't you? Just select the cells that you want to use as your data ranges, click on the Insert Chart button and follow the instructions – nothing could be simpler. But it's a wee bit time-consuming, isn't it? I think that you'll agree that this is another candidate for a bit of automation. Obviously we're going to be creating a new subroutine to create the chart for us, but before that we just need to make some temporary changes to the existing code.

First we'll stop the document from closing in the **download_stock_price** subroutine:

```
REM oDoc.close(true)
```

Then we'll stop the document from being hidden in **open_csv_file** with

```
oProperty(1).Name = "Hidden"
oProperty(1).Value = False
```

This means that we can develop the new code without having to open up the saved file to see what effect we've had. However, if you do this you must remember to enable the document close and document hiding functions once you've finished. If you're happy with that, create a new subroutine:

```
Sub insert_chart (doc as Object, cTitle as String)

End Sub
```

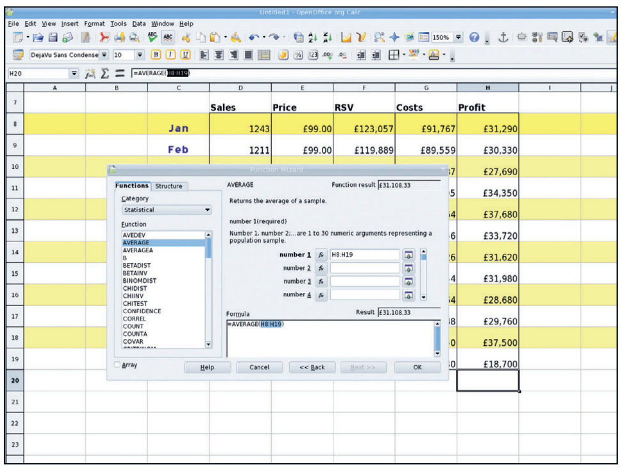
and then insert a new line into **download_stock_price**:

```
insert_chart (oDoc, "My Shares")
```

You'll need to place this just before the **save_file_as_ods** statement. Next we'll need to think about the code we need to add to our new subroutine.

If you've already tried manually creating a chart from the downloaded data, you'll realise that there's a problem: the key doesn't have a sensible title (it just says Column B). This is, of course, because the key comes from the first row – but in this case it's all data. To rectify this, we'll get the macro to insert a title for us:

```
Sub insert_chart (doc as Object, cTitle as String )
```


Untitled1 - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
DejaVu Sans Condense 10
150%
Function Wizard
Help Cancel << Back Next >> OK

H20 =AVERAGE(H8:H19)

	A	B	C	D	E	F	G	H
7				Sales	Price	RSV	Costs	Profit
8			Jan	1243	£99.00	£123,057	£91,767	£31,290
9			Feb	1211	£99.00	£119,889	£89,559	£30,330
10							7	£27,690
11							5	£34,350
12							4	£37,680
13							6	£33,720
14							6	£31,620
15							4	£31,980
16							4	£28,680
17							8	£29,760
18							0	£37,500
19							0	£18,700

Functions | Structure
Category: Statistical
Function: AVEDEV, AVERAGE, AVERAGEA, B, BETADIST, BETAINV, BINOMDIST, CHIDIST, CHIINV, CHITEST, CONFIDENCE, CORREL, COUNT, COUNTA, COVAR
AVERAGE — Function result £31,108.33
Returns the average of a sample.
number 1(required)
Number 1, number 2:...are 1 to 30 numeric arguments representing a population sample.
number 1: H8:H19
number 2:
number 3:
number 4:
Formula: =AVERAGE(H8:H19) — Result £31,108.33
Array

› If you need to make the figures in this *Calc* project clearer and more visually attractive, turn back to p34 to find out how.

```
Dim oSheet as Object, oCell as Object
oSheet = doc.Sheets(0)
oSheet.getRows.insertByIndex(0,1)
oCell = oSheet.getCellByPosition(1,0)
oCell.String = "Share Value"
End Sub
```

Just to explain what's going on here – we're selecting the first sheet of the spreadsheet, then we're inserting a new row, and then setting the contents of B1 to be Share Value.

If you now run Main and then create a chart manually, you'll see that the key now says Share Value. That's a good start: now let's get the macro to create the chart itself. The first thing that we need to do is get the macro to work out the dimensions of the data to be used:

```
r = 1
oCell = oSheet.getCellByPosition(0,r)
while oCell.String <> ""
 r = r + 1
 oCell = oSheet.getCellByPosition(0,r)
wend
```

With this information we need to define the range for the data in the sheet:

```
Dim oRange as Object
Dim oRangeAddress as Object

oRange = oSheet.getCellRangeByPosition(0,0,1,r – 1)
oRangeAddress = oRange.getRangeAddress
```

Now we can use this range to define the x and y data in a CellRangeAddress object:

```
Dim oCellRangeAddress(1) as New com.sun.star.table.
CellRangeAddress
'Set X axis
oCellRangeAddress(0).Sheet = oRangeAddress.Sheet
oCellRangeAddress(0).startColumn = oRangeAddress.
endColumn
oCellRangeAddress(0).endColumn = oRangeAddress.endColumn
oCellRangeAddress(0).startRow = oRangeAddress.startRow
oCellRangeAddress(0).endRow = oRangeAddress.endRow
'Set Y axis
oCellRangeAddress(1).Sheet = oRangeAddress.Sheet
oCellRangeAddress(1).startColumn = oRangeAddress.
startColumn
oCellRangeAddress(1).endColumn = oRangeAddress.
startColumn
oCellRangeAddress(1).startRow = oRangeAddress.startRow
oCellRangeAddress(1).endRow = oRangeAddress.endRow
```

The next step is to define the area for the chart. You'll find that the default size is quite small, so we just need to make it a little bigger:

```
Dim oRect as New com.sun.star.awt.Rectangle
oRect.Width= 20000
oRect.Height = 10000
```

Right, now we can write the code that will create the chart itself:

```
Dim oCharts as Object
oCharts = oSheet.Charts
oCharts.addNewByName(cTitle, oRect, oCellRangeAddress(),TR
UE,TRUE)
```

At this point you can run the macro if you want – and the chart will be created. However, there's still some tidying up to do – such as setting the chart title, and putting labels on the X and Y axis:

```
Dim oChart as Object
oChart = oCharts.getByName(cTitle).embeddedObject
oChart.HasMainTitle = True
oChart.Title.String = cTitle

oChart.diagram.HasXAxisTitle = True
oChart.diagram.XAxisTitle.String = "Company Symbol"

oChart.diagram.HasYAxisTitle = True
oChart.diagram.YAxisTitle.String = "Closing Value"
```

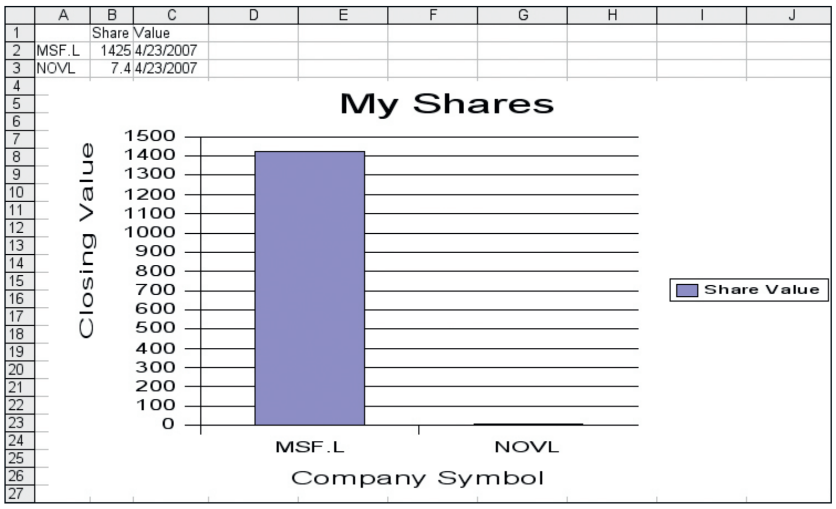
With that done, you've got a completely automated system for downloading your stock market data and creating a chart out of it. All you have to do now is re-enable the automatic closing and screen hiding… and then come in tomorrow morning, have a cup of coffee and see the results of your hard labour.

"Hang on!" I hear you cry ,"I don't want bar charts, I want the data in a doughnut – how do I do that?" Actually this is quite easy, but it's not very obvious from what we've done so far. We've only used the default chart style, and to use any other we have to use

```
oChart.diagram = oChart.createInstance("com.sun.star.chart.
DonutDiagram")
```

The complete list of charts that you can use is: AreaDiagram, BarDiagram (and that, as we've seen, is the default), DonutDiagram, LineDiagram, NetDiagram, PieDiagram, StackableDiagram, StockDiagram and XYDiagram.

Where you go from here is up to you. You can now download a CSV file directly into *Calc*, you can create any chart you want from the data, and you can do that all completely automatically without having to have any manual input at all. In this tutorial we've concentrated on Yahoo Finance because it's freely available and is in CSV format. You can, of course, use any CSV file, or in fact any data source – provided that you can load the data into *OOo* – so you could (for example) use this technique with a database as your source. However, if you do use this for analysing your stock market performance, and you do make a bundle, just remember who first pointed you in the right direction! LXF



	A	B	C
1		Share	Value
2	MSF.L	1425	4/23/2007
3	NOVL	7.4	4/23/2007

› With the minimum of effort you'll have a pretty chart showing you just how badly your shares are doing. Hurrah!