Lescasse Consulting

Getting Started with LC.Charts

January 12, 2015
Last Revised August 16, 2015

by Eric Lescasse

Lescasse Consulting
18 rue de la Belle Feuille
92100 Boulogne
France
Tel : (33) 6 47 98 13 61
eric@lescasse.com
http://www.lescasse.com

Table of Contents

Introduction

The purpose of **LC.Charts.dll** is to provide APL+Win Users with a simple and fast way to visualize their APL variables in charts and graphs.

The purpose is also to allow creating quality charts that can be copied to the clipboard and pasted in other application (Emails, Word, etc.), printed, saved to various images formats.

Things have been done so that it is as easy as it can possibly be to visualize an APL variable in a chart.

LC.Charts is a freeware and you are allowed to use it with no restriction, including in commercial applications or products.

Installation

WARNING: LC.Charts v2.5.0.0+ requires APL+Win v15.1

LC.Charts can be downloaded from:

http://www.lescasse.com/content/lccharts.aspx
(see: http://www.lescasse.com/content/lcchartshistory.aspx for details about the latest version!)

where the detailed installation instructions are described.

Basically, there is an LCChartsSetup.exe that you need to run: installation takes less than 30 seconds.

After installing the product, you need to manually set the LCCHARTS.SF User Command file as your top level User Command file.

Assuming your User Command Processor is working, i.e. you should be able to issue the following command in your APL Session without any error:

```
]ufile
```

you can set LCCHARTS as your top level User command file with the following command:

```
]ufile fullInstallPath\lccharts
```

i.e. if you installed LC.Charts in the **C:\Aplwin15\LCCharts** folder, you should issue the following command:

```
]ufile c:\aplwin15\lccharts\lccharts
```

Note that there is an Uninstaller that allows you to completely uninstall LC.Charts from your computer if you want or need to.

Important note (new in v2.3.0.0):

Note that LC.Charts does not come with any APL+Win workspace, but only with a User Command APL+Win file called LCCHARTS.SF.

Once you have installed LC.Charts as recommended on my Web site, and therefore set LCCHARTS as your top level User Command file, you can load all LC.Charts utilities in a CLEAR WS with the following simple command:

```
]uload *
```

Requirements

In order to use LC.Charts, you need:

1. To have the **.Net Framework 4.0+** installed on your computer[1]
2. To have **APL+Win v4.0+** installed on your computer
3. To have the **APL+Win User Command processor** up and running[2]

Supporting high DPI and Windows large fonts (new in v2.5.0.0)

Starting with v2.5.0.0, the LC.Charts.dll now supports higher DPI resolutions than 96 DPI as well as Windows large fonts.

However, to allow LC.Charts to work correctly with APL+Win in such high resolutions, you must add the following yellow highlighted lines to your APL+Win v15+ **aplw.exe.manifest** file:

```
<?xml version="1.0" encoding="UTF-8"?>
    <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
        <asmv3:application xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"> <!-- ELE5apr15 -->
            <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
                <dpiAware>true</dpiAware>
            </asmv3:windowsSettings>
        </asmv3:application>
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
        <security>
        …
```

These yellow highlighted lines must be added right below the <assembly …> tag line.

[1] You can check the versions of the .Net Framework that are installed on your computer by looking at the **C:\Windows\Microsoft.Net\Framework** and **C:\Windows\Microsoft.Net\Framework64** sub-folders: if you find a sub-folder called v4.0.30319, then you have the .Net Framework 4.0 installed on your computer

[2] For the APL+Win User Command Processor to work, you need to have a line reading:
UcmdFile={path}\UCMDS
in the **[Config]** section of your **APLW.INI** file, where **{path}** is the complete path of your **APLW.EXE** executable, and you need to have at least the **UCMDS.SF** file present in the same **{path}**. In case of any problem, please contact me.

Basics

There are 2 ways you can use LC.Charts from any workspace:

1. As a **]chart** User Command (because nothing is simpler to use than a User Command)
2. As an ActiveX through the use of ⎕wi

You should use the]chart command when you want a quick way to visualize APL variables from your APL Session.

You should probably use ⎕wi instead if you want to produce charts under program control (although you could also use ⎕ucmd']chart myVariable /option1 /option2=...')

Using the]chart user command

It is as simple as:

```
]chart ?20⍴20
```

Or:

```
data←(⍳10),?10 2⍴100000
]chart data
```

Using ⎕wi

Proceed as follows:

```
'ff'⎕wi'*Create' 'LC.Charts.Chart'
'ff'⎕wi'*xData'(?20⍴20)
'ff'⎕wi'*XShow'
```

Or:

```
data←(⍳10),?10 2⍴100000
'ff'⎕wi'*Create' 'LC.Charts.Chart'
'ff'⎕wi'*xData'data
'ff'⎕wi'*XShow'
```

In both cases, you can pass any APL expression or APL variable to LC.Charts as long as it returns a numeric vector or numeric matrix.

When you pass a numeric matrix, the first column should contain the X values and the remaining columns the Y values for your various series.

Closing the C# Chart Form (new in v2.5.0.0)

Whether using the **LC.Charts.Chart** or **LC.Charts.Chart3D** object, you can now close the C# form displaying the chart you have produced by simply pressing **Escape**.

Getting Documentation

You can use the following important]chart options to get help with the]chart command.

Find documentation about the]chart command itself:

```
      ]chart?
```

Find the **LC.Charts.dll** version you are using:

```
      ]chart /v
1.8.0.0
```

Find all the properties that can be used with ⎕wi:

```
      ]chart /props
xCaption            xPointLabels        xTopMost            xXMinorStep
xChartAxisTypes     xPointLabelsAngle   xType               xXTitleVisible
xChartColors        xPointLabelsFormat  xVersion            xYAngle
xChartSymbols       xReflect            xWhere              xYAxisTitle
xChartTypes         xSmooth             xXAngle             xYAxisType
xCrossHairCursor    xSymbol             xXAxisTitle         xYMajorStep
xCurveFill          xSymbolFill         xXAxisType          xYMax
xData               xSymbolSize         xXMajorStep         xYMin
xLegend             xTitle              xXMax               xYMinorStep
xLegendVisible      xTitleVisible       xXMin               xYTitleVisible
```

Find all the methods that can be used with ⎕wi:

```
      ]chart /meth
XClose   XRedraw  XShow    XWait
```

It is also possible to get detailed documentation about any of the above properties.

Please remember that LC.Charts properties should always be prefixed with a lowercase x when used with ⎕wi.

For example there is an LC.Charts property called **XAngle**; when you use it from APL you must specify it as **xXangle**.

Similarly, methods should always be prefixed with an uppercase X when used with ⎕wi.

The]chprops,]chmeths,]chevents user commands (new in v2.5.0.0)

The new]chprops,]chmeths and]chevents user commands let you retrieve properties, methods and events names for both the Chart and the Chart3D object.

They are very handy to quickly find out the property or method you want to use when you to not remember their names or remember only paret of their names.

```
      ]chprops
xAllowDragPoint         xOutsideColor             xXAxisMagAuto
xAllowZoom              xOverlay                  xXAxisStyle(3D)
xAltitudeStyle(3D)      xPerspective(3D)          xXAxisStyles(3D)
xAltitudeStyles(3D)     xPlanesStyle(3D)          xXAxisTitle
xAxisColor              xPlanesStyles(3D)         xXAxisTitle(3D)
xBarChartStyle(3D)      xPointLabels              xXAxisTitleFont
xBarChartStyles(3D)     xPointLabelsAngle         xXAxisType
xBottomMargin(3D)       xPointLabelsFormat        xXLabelFormat(3D)
…

      ]chmeths
XAddFootnote(3D)  XEuroDollarRates       XLocalizeTo      XShow
XAddNote(3D)      XGetDPI(3D)            XRedraw          XShow(3D)
XAddText          XGetScalingFactor      XRedraw(3D)      XWait
XClose            XGetScalingFactor(3D)  XSetMargins(3D)

      ]chevents
XMouseMoved(3D)
```

The properties, methods and events which refer to the Chart3D object are marked with (3D).

Additionally, you can specify an optional argument to narrow your search:

```
      ]chprops title
xSubTitle(3D)  xTitleVisible    xXTitleVisible    xYAxisTitle(3D)
xTitle         xXAxisTitle      xY2AxisTitle      xYAxisTitleFont
xTitle(3D)     xXAxisTitle(3D)  xY2AxisTitleFont  xYTitleVisible
xTitleFont     xXAxisTitleFont  xYAxisTitle       xZAxisTitle(3D)

      ]chprops title(3d)
xSubTitle(3D)  xXAxisTitle(3D)  xZAxisTitle(3D)
xTitle(3D)     xYAxisTitle(3D)
```

These user commands are case insensitive.

The]chdoc user command (new in v2.5.0.0)

You can get documentation on any property or method by using the new]chdoc user command.

Just provide the full name of the property or method **without providing the x or X prefix** (note: the user command is case insensitive):

```
      ]chdoc ylabelformat
xYLabelFormat(3D) documentation
-------------------------------
The xYLabelFormat property lets you get or set the format for Y-Axis labels

Syntax:   ylabelformat←'ff.cc'⎕wi'*xYLabelFormat'
          'ff.cc'⎕wi'*xYLabelFormat'ylabelformat

ylabelformat:  a .Net compatible formaty string specification

Note:
Look at:  http://www.cheat-sheets.org/saved-copy/msnet-formatting-strings.pdf
for a cheat sheet about .Net format strings.  Look particularly at the
"Custom Numeric Format Strings Output Examples" at the bottom right of
the page are these are easy to understand and the ones you most likely
want to use.

Example:
      SampleCurves ⋄ ⎕wi'*xReflect'1 ⋄ ⎕wi'*xTopMost'1
      ⎕wi'*xScale'.8 ⋄ ⎕wi'*XRedraw'
      ⎕wi'*xYLabelFormat' '#.00'

See also:
      XLabelFormat
      ZLabelFormat
```

If the property or method exist for both the Chart and the Chart3D objects, then both documentation are retrieved. Example:

```
      ]chdoc xangle
xangle documentation
--------------------
The xXAngle property lets you get or set the angle for the X-Axis labels

Syntax:    xangle←'ff'⎕wi'*xXAngle'
           'ff'⎕wi'*xXAngle'xangle
```

```
          ]chart data /xa=xangle
          ]chart data /xa=⍎xangleVar

xangle:    an integer scalar representing the X-axis labels angle (default it 0)
xangleVar: an APL variable name containing the xangle value

Example:
      ]chart ((⍳10),?10 2⍴100000) /xa=30
      ]chart ((⍳10),?10 2⍴100000) /xa=90

      'ff'⎕wi'*Create' 'LC.Charts.Chart'('xReflect'1)('xTopMost'1)
      'ff'⎕wi'*xData'((⍳10),?10 2⍴100000)
      'ff'⎕wi'*XShow'
      'ff'⎕wi'*xXAngle'30
      'ff'⎕wi'*xXAngle'60
      'ff'⎕wi'*xXAngle'90
      'ff'⎕wi'*xXAngle'330

See Also:
xYAngle


xangle(3D) documentation
------------------------
The xXAngle property lets you get or set the angle with which the X-Labels are
drawn

Syntax:   xangle←'ff.cc'⎕wi'*xXAngle'
          'ff.cc'⎕wi'*xXAngle'xangle

xangle:  an numeric scalar between 0 and 360

Example:
      SampleCurves ⋄ ⎕wi'*xReflect'1 ⋄ ⎕wi'*xTopMost'1
      ⎕wi'*xScale'0.8 ⋄ ⎕wi'*XRedraw'
      ⎕wi'*xXAngle'90
      ⎕wi'*xXAngle'30
      ⎕wi'*xXAngle'45

See also:
```

Getting detailed documentation about any property:

You can also retrieve documentation using the]chart /doc= or]chart3d /doc= user commands, but in that case you must provide the property or method name with its x (or X) prefix and using the right case.

Example:

```
      ]chart /doc=xXAngle
The xXAngle property lets you get or set the angle for the X-Axis labels

Syntax:    xangle←'ff'⎕wi'*xXAngle'
           'ff'⎕wi'*xXAngle'xangle
           ]chart data /xangle=xangle
           ]chart data /xangle=⍎xangleVar
           (smallest abbreviation: /xa= )

xangle:    an integer scalar representing the X-axis labels angle (default it 0)
xangleVar: an APL variable name containing the xangle value

Example:
      ]chart ((⍳10),?10 2⍴100000) /xa=30
      ]chart ((⍳10),?10 2⍴100000) /xa=90

      'ff'⎕wi'*Create' 'LC.Charts.Chart'('xReflect'1)('xTopMost'1)
      'ff'⎕wi'*xData'((⍳10),?10 2⍴100000)
      'ff'⎕wi'*XShow'
      'ff'⎕wi'*xXAngle'30
      'ff'⎕wi'*xXAngle'60
      'ff'⎕wi'*xXAngle'90
      'ff'⎕wi'*xXAngle'330

See Also:
xYAngle
```

The detailed properties documentation always end with reproducible examples, so that you can put your cursor on these example lines in the APL Session and press Enter to execute them.

This is a good way to see how properties work and learn how to use LC.Charts.

It is also easy to experiment with changing the values used in the examples before hitting Enter.

Using]chart Options

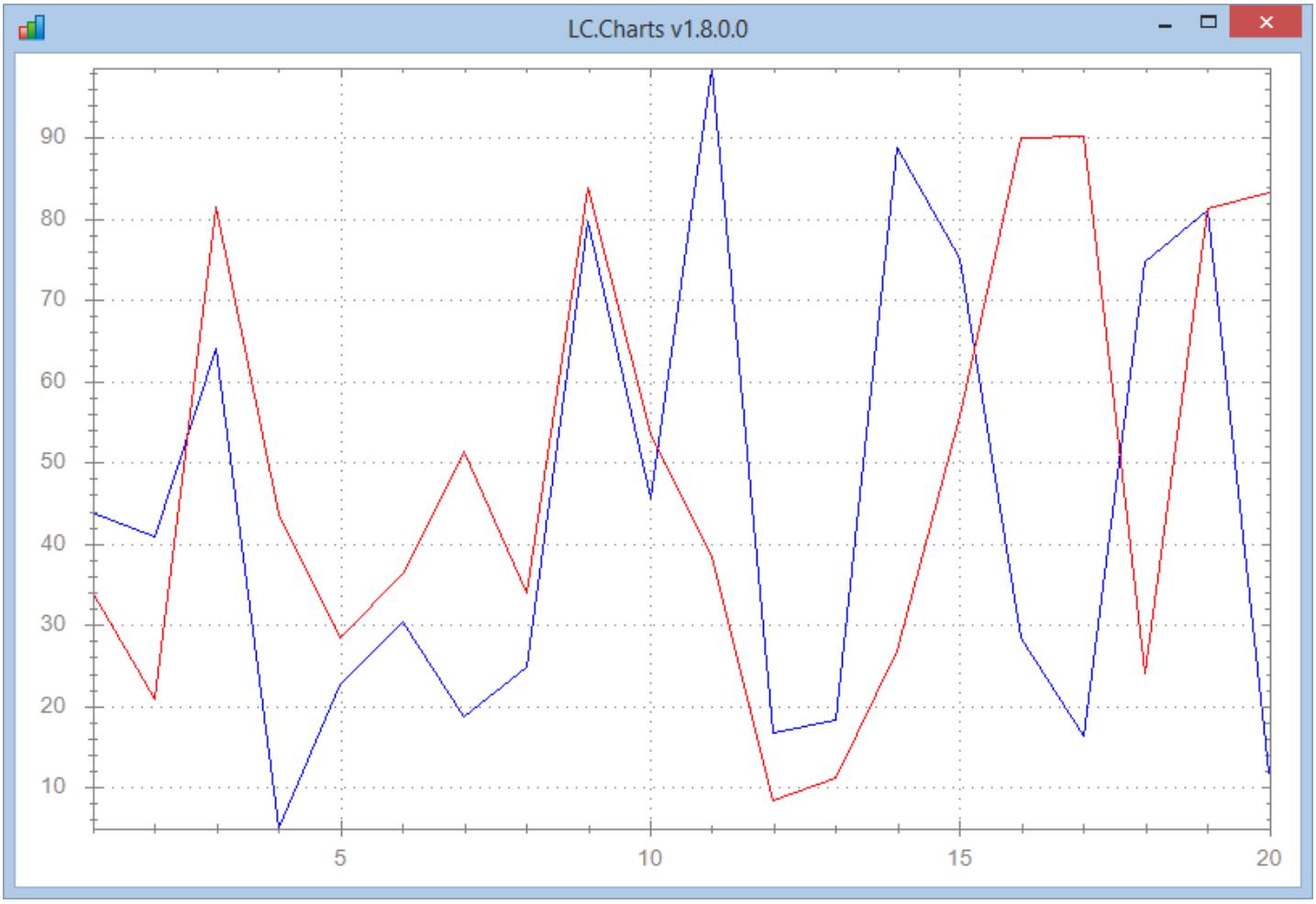
By default, when you simply use:

```
]chart myVariable
```

LC.Charts displays an as simple chart as possible with just your curve(s) and axis.

Example:

```
]chart (⍳20),?20 2⍴100000
```



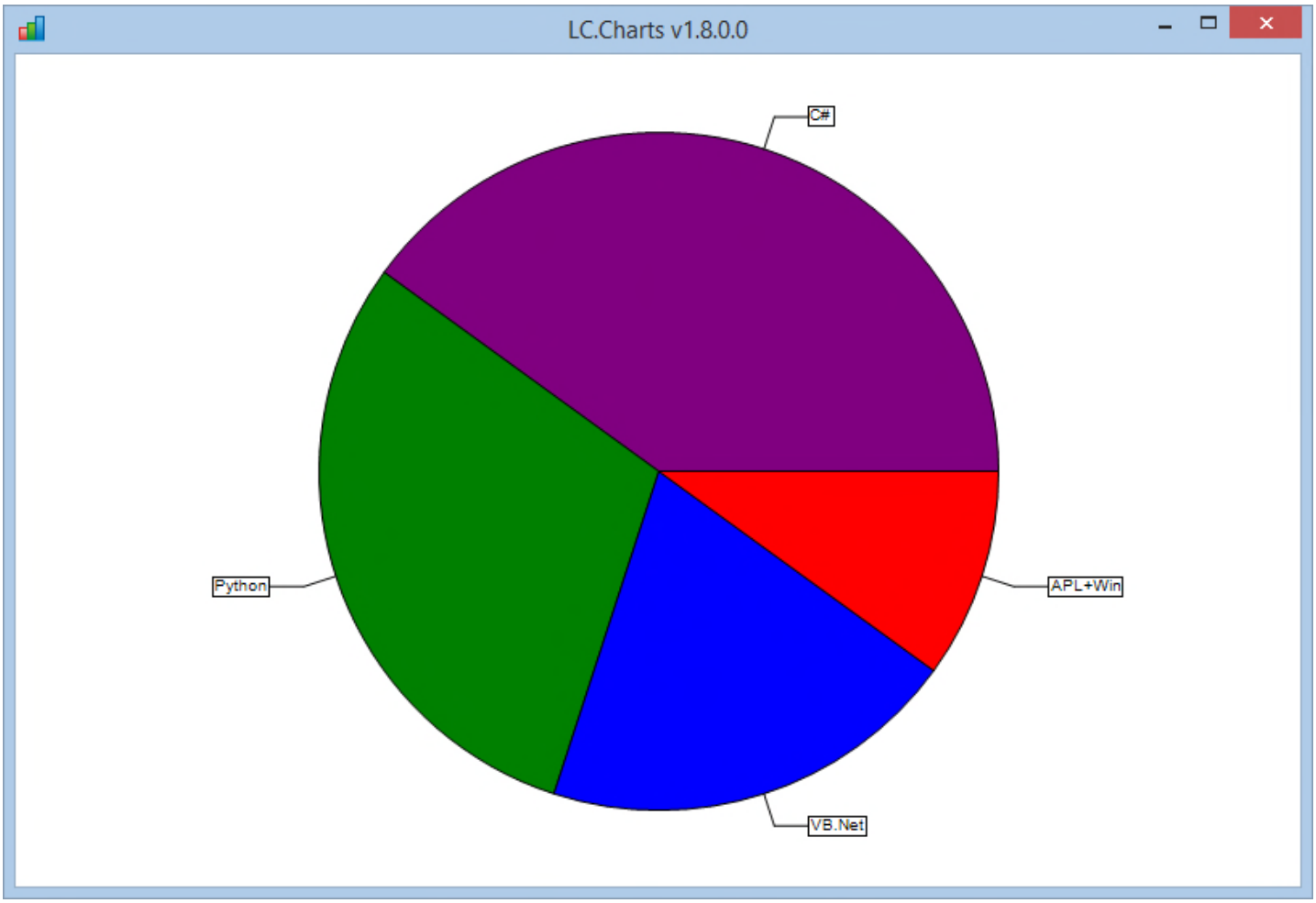
The]chart command has a large set of options you can use at any time to customize your chart.

The most important of these options is the /type option which lets you choose the type of charts you want to use to display your variable.

The possible choices are: bar, curve, pie, scatter

Example:

```
]chart (10 20 30 40)('APL+Win' 'VB.Net' 'Python' 'C#') /type=pie
```



To get documentation about the]chart user command simply do:

```
]chart?
```

The possible options, as of LC.Charts version 2.0, are:

```
]chart data {/caption} {/cf} {/chc} {/doc=} {/legend=} {/pl}
           {/plangle=} {/plformat=} {/methods} {/properties}
           {/smooth=} {/snofill} {/symb=} {/ssize=} {/title=}
           {/topmost} {/type=} {/version}
           {/xangle=} {/xmajstep=} {/xmax=} {/xmin=} {/xminstep=} {/xtitle=} {/xtype=}
           {/yangle=} {/ymajstep=} {/ymax=} {/ymin=} {/yminstep=} {/ytitle=} {/ytype=}
```

When you use an option you don’t need to type the entire option name: you must type at least the minimum set of characters that allow to distinguish it from other LC.Charts options.

For example, you can use:

```
]chart myVariable /xmajstep=10000
```

or:

```
]chart myVariable /xmaj=10000
```

but not:

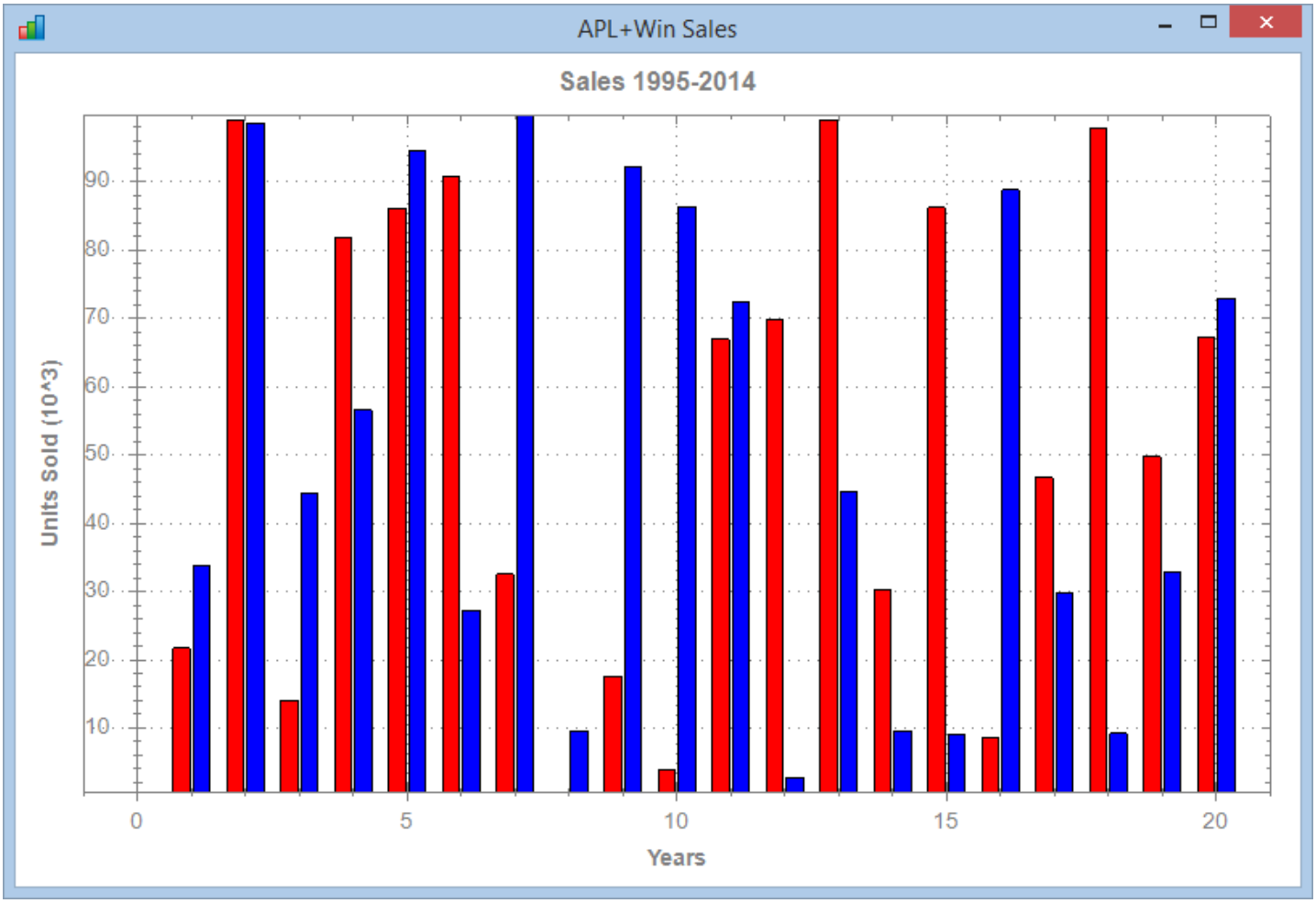
```
]chart myVariable /xma=10000
```

because there is another option called /xmax and the system would not be able to determine if you wanted to use /xmax or /xmajstep.

The order in which you specify options is irrelevant.

Here is the same chart as above using a few options:

```
      aplSales←(⍳20),?20 2⍴100000
      ]chart aplSales /ca=APL+Win Sales /ti=Sales 1995-2014 /xti=Years /yti=Units
Sold /ty=bar /xmin=-1 /xmax=21
```

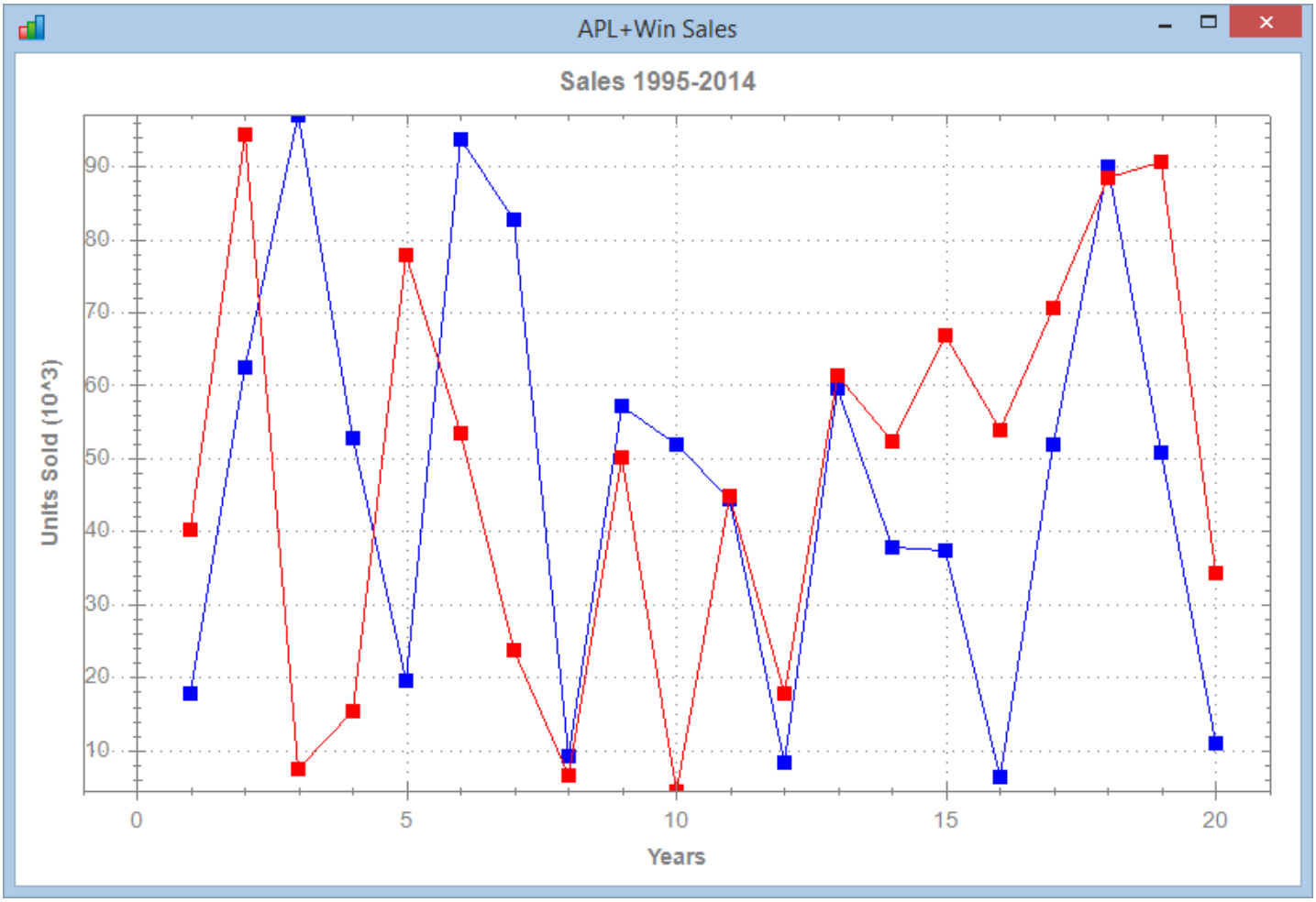


Using Dynamic Options

To make it easier to use the]chart command under program control with ⎕ucmd, you can also assign your option values to APL variables and use these variables prefixed by ⍎ when you specify]chart options.

Here is an example:

```
      aplSales←(⍳20),?20 2⍴100000
      caption←'APL+Win Sales'
      title←'Sales 1995-2014'
      xaxistitle←'Years'
      xmin←¯1
      xmax←21
      ]chart aplSales /ty=curve /ca=⍎caption /ti=⍎title /xti=⍎xaxistitle
/yti=Units Sold /xmin=⍎xmin /xmax=⍎xmax /sy=square
```



Using LC.Charts with ⎕wi

You can use LC.Charts with ⎕wi as you would use any other ActiveX from APL+Win.

Creating an instance of the LC.Charts.Chart object

You must first create an instance of the LC.Charts.Chart object with:

```
'anyname'⎕wi'*Create' 'LC.Charts.Chart'
```

Note that, by default, at this stage, the LC.Charts form is not made visible.

Supplying data to the LC.Charts.Chart object

You must then supply data to be drawn to the chart object using the xData property:

```
'anyname'⎕wi'*xData'(?10⍴100)
```

Note that it is the xData property that effectively draws the chart. However, at this stage, the LC.Charts window is still not yet visible.

Showing the chart

Finally you need to call the XShow method to display the LC.Charts window.

```
'anyname'⎕wi'*XShow'
```

Using more properties

You can then use more LC.Charts properties to customize your chart:

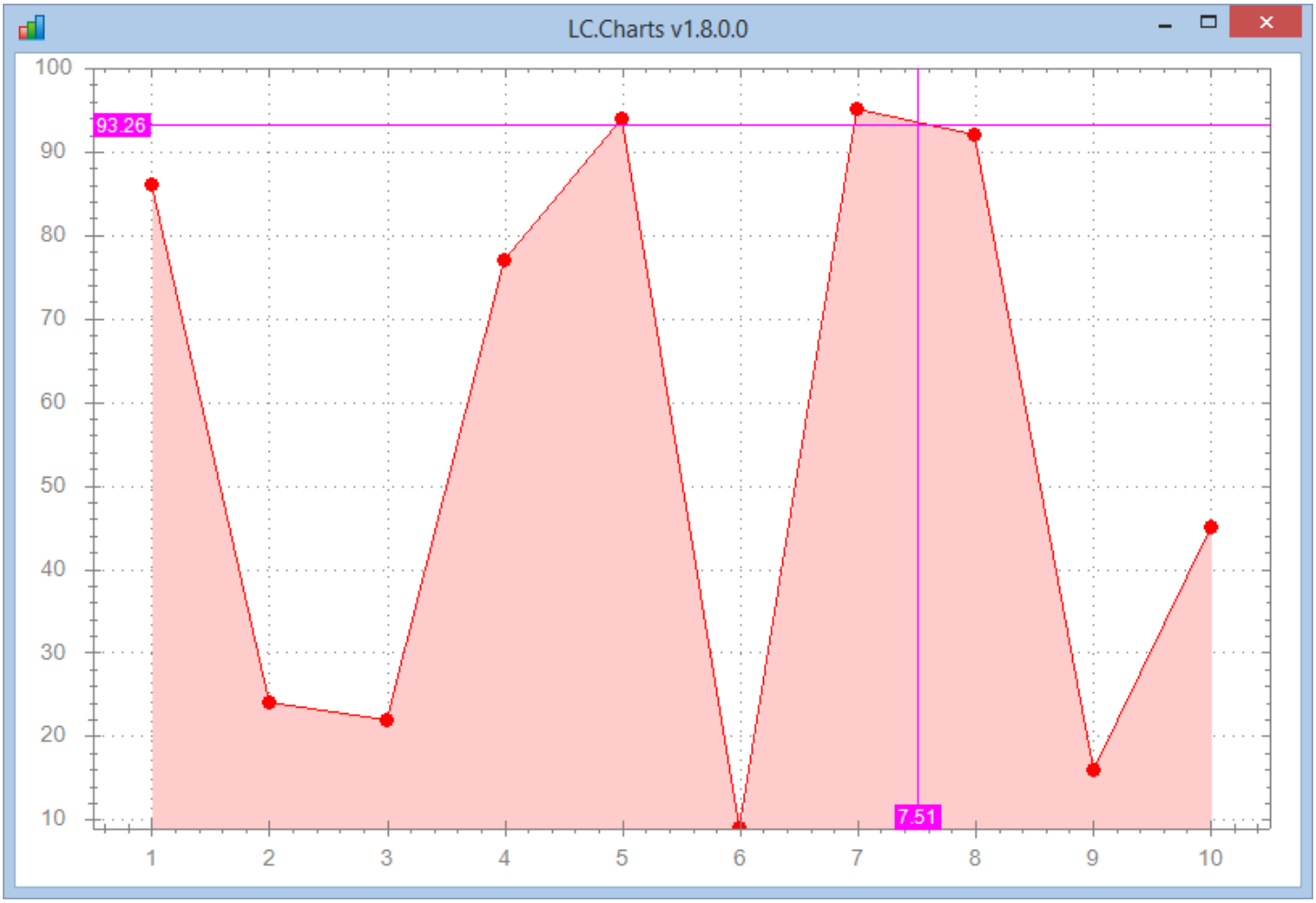
```
'anyname'⎕wi'*xCurveFill'80
'anyname'⎕wi'*xSymbol' 'circle'
'anyname'⎕wi'*xXMin'0.5
'anyname'⎕wi'*xXMax'10.5
'anyname'⎕wi'*xYMax'100
'anyname'⎕wi'*xCrossHairCursor'1
```

You may be surprised that setting the above properties does not seem to have any effect on the chart you have drawn!

Using the XRedraw method

This is, by design, to avoid the chart being redrawn every time you change a property, so setting properties is delayed until you call the XRedraw method:

```
'anyname'□wi'*XRedraw'
```



You can remember that:

1. **xData** is the property that draws the chart
2. If you set any property after having set **xData**, you will need to call **XRedraw** to redraw the chart

The xTopMost and xReflect properties

Two LC.Charts properties are invaluable when you want to explore the LC.Charts possibilities.

The first one, xTopMost, allows you to make the LC.Charts form topmost:

```
'anyname'⎕wi'*xTopMost'1
```

The second one, xReflect, allows you to immediately see chart changes when setting a property (without having to call XRedraw):

```
'anyname'⎕wi'*xReflect'1
```

Therefore, when working in the APL Session and exploring LC.Charts, it is recommended to do:

```
'anyname'⎕wi'*xTopMost'1
'anyname'⎕wi'*xReflect'1
```

The Where property

Additionally, the **xWhere** property is similar to the APL **where** property and allows you to change the LC.Charts window position and size (note that coordinates are pixels):

```
'anyname'⎕wi'*xWhere'0 0
```

```
'anyname'⎕wi'*xWhere'300 400 500 600
```

Closing the LC.Charts window

You can of course manually close the LC.Charts window by clicking its Close button.

You can also programmatically close it using the **XClose** method.

```
'anyname'⎕wi'*XClose'
```

Note that using:

```
      'ff'⎕wi'*Create' 'LC.Charts.Chart'('*xData'(⍳10))'*XShow'
ff
      'ff'⎕wi'*Create' 'LC.Charts.Chart'('*xData'(⍳10))'*XShow'
ff
```

creates 2 LC.Charts windows, i.e. calling '*Create' to create an instance DOES NOT destroy the previous instance(s) with the same name.

Querying Properties and Methods

Just as for any other wi object, you can query the available object properties at any time:

```
      'ff'wi'*properties'
apldata children class clsid data def description errorcode errormessage events
       instance interface links methods modified modifystop name obj opened pend
      ing progid properties self state suppress unicodebstr version xAllowDragPo
      int xAllowZoom xAxisColor xCaption xChartAxisTypes xChartColors xChartSymb
      ols xChartTypes xCrossHairCursor xCurveFill xData xFontsScaled xHandle xIn
      sideColor xLegend xLegendBorder xLegendFont xLegendPosition xLegendPositio
      ns xLegendVisible xLineColor xLineStyle xLineStyles xLineWidth xOutsideCol
      or xOverlay xPointLabels xPointLabelsAngle xPointLabelsFormat xReflect xSm
      ooth xSymbol xSymbolFill xSymbolSize xTitle xTitleFont xTitleVisible xTopM
      ost xType xUseY2Axis xVersion xWhere xXAngle xXAxisFormat xXAxisLabelsFont
       xXAxisTitle xXAxisTitleFont xXAxisType xXMajorStep xXMax xXMin xXMinorSte
      p xXTitleVisible xY2AxisFormat xY2AxisLabelsFont xY2AxisTitle xY2AxisTitle
      Font xY2MajorStep xY2Max xY2Min xY2MinorStep xYAngle xYAxisAtX xYAxisForma
      t xYAxisLabelsFont xYAxisTitle xYAxisTitleFont xYAxisType xYMajorStep xYMa
      x xYMin xYMinorStep xYTitleVisible

      'ff'wi'*methods'
Close Create Defer Delete EnumEnd EnumNext EnumStart Event Exec Info Modify New
       Open Ref Send Set SetLinks XAddText XAllowDragPointDoc XCaptionDoc XChart
      AxisTypesDoc XChartColorsDoc XChartSymbolsDoc XChartTypesDoc XClose XCross
      HairCursorDoc XCurveFillDoc XDataDoc XEuroDollarRates XHandleDoc XLegendDo
      c XLegendVisibleDoc XLocalizeTo XLocalizeToDoc XOverlayDoc XPointLabelsAng
      leDoc XPointLabelsDoc XPointLabelsFormatDoc XRedraw XReflectDoc XShow XSmo
      othDoc XSymbolDoc XSymbolFillDoc XSymbolSizeDoc XTitleDoc XTitleVisibleDoc
       XTopMostDoc XTypeDoc XVersionDoc XWait XWhereDoc XXAngleDoc XXAxisTitleDo
      c XXAxisTypeDoc XXMajorStepDoc XXMaxDoc XXMinDoc XXMinorStepDoc XXTitleVis
      ibleDoc XYAngleDoc XYAxisAtXDoc XYAxisTitleDoc XYAxisTypeDoc XYMaxDoc XYMi
      nDoc XYTitleVisibleDoc
```

Note that most methods are documentation methods.

Getting Property Documentation

So, once you have created an instance of LC.Charts.Chart, you can get documentation on a given property by simply calling its documentation method.

Example:

```
      'ff'⎕wi'*XCurveFillDoc'
The xCurveFill property lets you get or set whether the area below curves in a curve-type
chart are painted or not and to indicate the level of paint color intensity.

Syntax:    curveFill←'ff'⎕wi'*xCurveFill'
           'ff'⎕wi'*xCurveFill'curveFill
           ]chart data /cf=curveFill
           (smallest abbreviation:  /cf= )

curveFill:  an integer between -99 and 99
            Values higher than 0 make the paint color lighter than the curve color
            Values less than 0 make the paint color darker than the curve

Notes:
1. Setting xCurveFill to 0 is the same as not painting areas below curves
2. Point labels can't be displayed when you set xCurveFill to a value different from 0
3. The symbol interior fill color is forced to white if you set xCurveFill to a value
   less than 60 (the reason is that otherwise, it gets hard to see point symbols)

Example:
      'ff'⎕wi'*xCurveFill'
0
      'ff'⎕wi'*xCurveFill'80
```

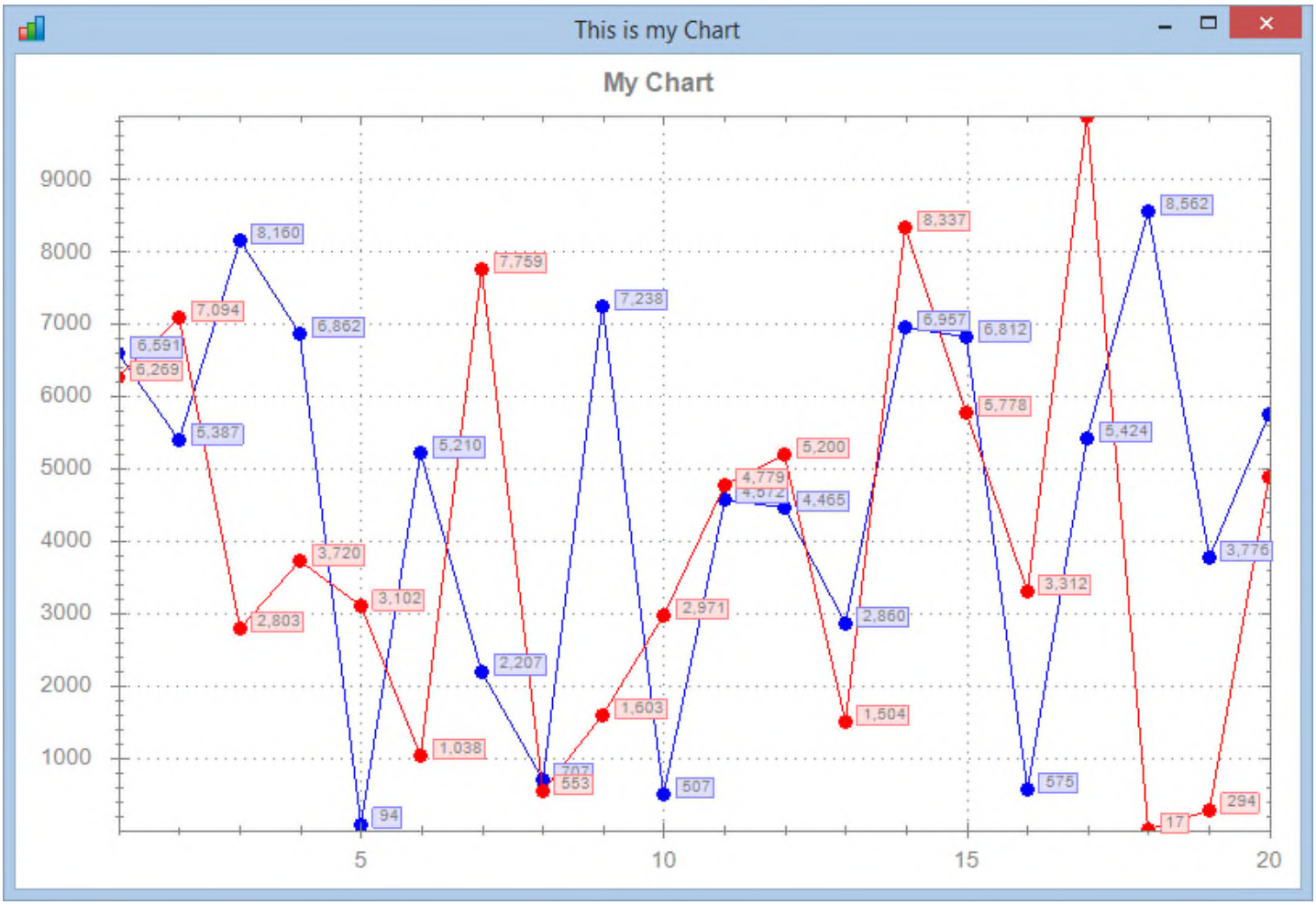
Combining use of]chart and use of □wi

Note that you can perfectly combine the use of]chart and □wi to customize your chart. All you need to know is that the LC.Charts instance created by]chart is named: **lccharts**

In that case, start by create a chart using]chart:

```
]chart (⍳20),?20 2⍴10000 /chc /pl

'lccharts'□wi'*Set'('*xTopMost'1)('*xReflect'1)
'lccharts'□wi'*xPointLabelsFormat' '#,###'
'lccharts'□wi'*xSymbol' 'circle'
'lccharts'□wi'*xTitle' 'My Chart'
'lccharts'□wi'*xCaption' 'This is my Chart'
```



Customizing your chart (new in v2.3.0.0)

LC.Charts v2.3.0.0 introduces a good number of new properties to let you customize your chart.

Setting Colors (new in v2.3.0.0)

You can set various colors:

- The **xInsideColor** property to set the chart color (inside the chart axis)
- The **xOutsideColor** property to set the chart color (outside the chart axis)
- The **xAxisColor** property to change th chart axis color
- The **xLegendBorder** property to set the legend fill color
- The **xLineColor** to set the various curve colors for a curve chart

These colors can be defined as plain solid colors or as gradients.

To define a plain solid color, use just one RGB set of values:

```
'ff'□wi'*xOutsideColor'(64 64 255)
```

To define a gradient color, use 2 RGB set of values (the start color and the end color of the gradient) and use and angle in degrees to indicate the gradient direction. Example:

```
'ff'□wi'*xInsideColor'(140 140 255)(224 224 255)90
```

Line and axis colors can only be defined as plain solid colors.

Fill colors can be defined either as plain solid colors or gradient colors.

Some properties like **xLegendBorder** contain other information than just color.

```
'ff'□wi'*xLegendBorder'1(0 0 192)((255 255 0)(255 192 192)45)
```

1 indicates whether the legend should have a border (1) or not (0)
(0 0 192) indicates the legend border color
((255 255 0)(255 192 192)45) is a gradient color for the legend fill color

Setting Fonts (new in v2.3.0.0)

You can set various fonts:

- Use **xTitleFont** to set the general title font
- Use **xLegendFont** to set the legend font
- Use **xXAxisTitleFont** to set the X-Axis title font
- Use **xYAxisTitleFont** to set the Y-Axis title font
- Use **xY2AxisTitleFont** to set the secondary Y-Axis title font
- Use **xXAxisLabelsFont** to set the X-Axis labels font

- Use **xYAxisLabelsFont** to set the X-Axis labels font
- Use **xY2AxisLabelsFont** to set the X-Axis labels font

Fonts should be specified using a nested vector containing:

(fontname) (fontsize) (fontstyle) (fontcolor)

- **fontname** should be a font family name (i.e. 'Arial' or 'Times New Roman' or 'Calibri', …)
- **fontsize** should be the font size expressed in pixels
- **fontstyle** should be the sum of 1=bold, 2=italic and/or 4=underline
- **fontcolor** should be a 3-element RGB integer vector

Example:

```
      'ff'⎕wi'*xLegendFont' 'Helvetica'12 3(0 0 192)
```

Customizing the Legend (new in v2.3.0.0)

You can use the following properties to customize the legend:

- **xLegendPosition** to position the legend almost anywhere you want in the chart
- **xLegendPositions** to know which value you can use for the **xLegendPosition** property
- **xLegendFont** to change the legend text font and color
- **xLegendBorder** to add a border around the legend and paint the legend

Example:

```
      80 TELPRINT⊃'ff'⎕wi'*xLegendPositions'
Bottom           InsideBotLeft    Left             TopFlushLeft
BottomCenter     InsideBotRight   Right
BottomFlushLeft  InsideTopLeft    Top
Float            InsideTopRight   TopCenter
      'ff'⎕wi'*xLegendPosition' 'Left'
      'ff'⎕wi'*xLegendFont' 'Helvetica'12 3(0 0 192)
      'ff'⎕wi'*xLegendBorder'1(0 0 192)((255 255 0)(255 192 192)45)
```

Customizing the Curves in a curve chart (new in v2.3.0.0)

You may customize each curve in your chart by setting the following properties (with one value per curve in the chart)

- **xLineWidth** sets the curve width for each curve
- **xLineStyle** sets the curve style for each curve
- **xLineStyles** returns the possible values you can use for the **xLineStyle** property
- **xLineColor** allows you to set a color for each curve

Example:

```
      80 TELPRINT⊃'ff'⎕wi'*xLineStyles'
0=Solid[default]  2=DashDot         4=Dot
1=Dash            3=DashDotDot
      'ff'⎕wi'*xLineWidth'3 2 2
      'ff'⎕wi'*xLineStyle'0 1 4   ⍝ 0=solid 1=Dash 2=DashDot 3=DashDotDot 4=Dot
      'ff'⎕wi'*xLineColor'(255 0 0)(0 128 0)(0 0 192)
```

Creating multi-line titles (new in v2.3.0.0)

With the LC.Charts.Chart object you can create multi-line title.

To do so, just insert ⎕tclf characters to separate the various lines.

Example:

```
      'ff'⎕wi'*xTitle'('Fictitious € to USD,GBP,AUD Conversion
Rates',⎕tclf,'(since the beginning of Euro)')
```

Note that this should work for any text you specify through LC.Charts titles, labels, text properties.

Disabling automatic magnitude adjustments (new in v2.3.0.0)

The xMagAuto, xXAxisMagAuto, xYAxisMagAuto and xY2AxisMagAuto properties (new in v2.3.0.0)

By default LC.Charts.Chart auto-adjust the axis label values so that they display reasonable numbers.

For example, without magnitude auto-adjust the **Brent10** APL function would display the following Y label:

50000000000000000000000000000

because the largest ordinate in the chart is 8.569854E31.

So in such a case, automatic magnitude adjustment is clearly a good thing.

However, in many cases, you will like to see real non adjusted X and Y label values.

To achieve that you can set the following properties to 0:

- **xAutoMag** (same as setting **xXAxisAutoMag**, **xYAxisAutoMag** and **xY2AxisAutoMag** to 0)
- **xXAxisAutoMag**
- **xYAxisAutoMag**
- **xY2AxisAutoMag**

The /nomagauto]chart option (new in v2.3.0.0)

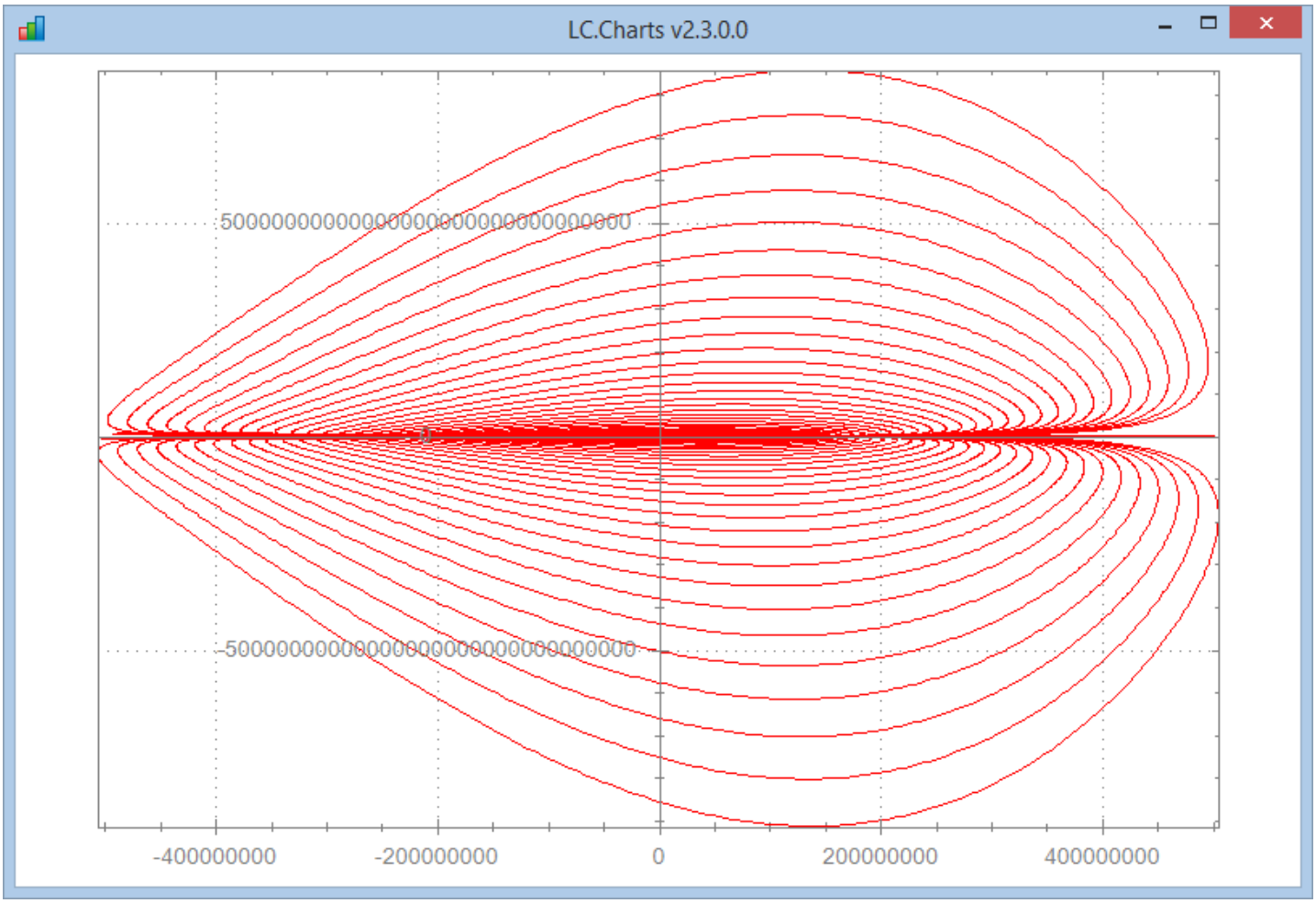
If you use the **]chart** command, you can use the **/nomagauto** or **/nomag** option which is equivalent to setting **xMagAuto** to **0**.

Example:

Change the Brent10 function to the following:

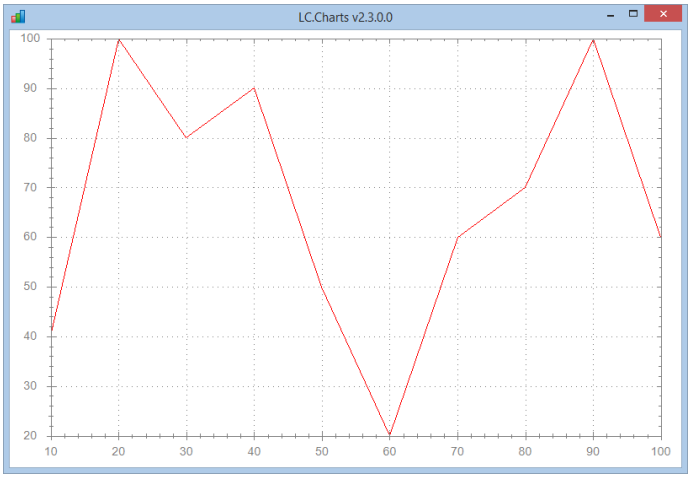
```
    ∇ Brent10;a;r;x;y;z
[1]   ⍝∇ Brent10 -- Function nicely provided by Brent Hildebrand
[2]
[3]   a←0 19800 Step .1
[4]   r←a+a×Sin 4×a
[5]   x←r×a×Cos 4×a
[6]   y←(r×Sin a)×7
[7]   ⎕ucmd'chart x,[1.1]y /nomag'
[8]
    ∇
```

Running Brent10 will display the following chart:

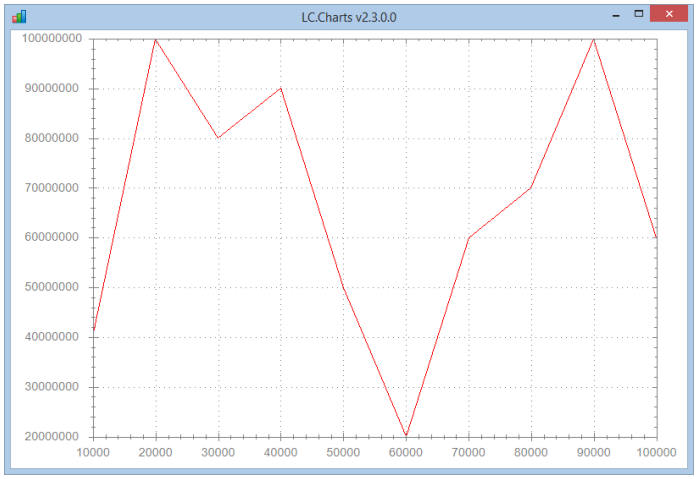


Here are some more examples:

```
data←(10000×⍳10),[1.5]10000000×?10⍴10
]chart data
```



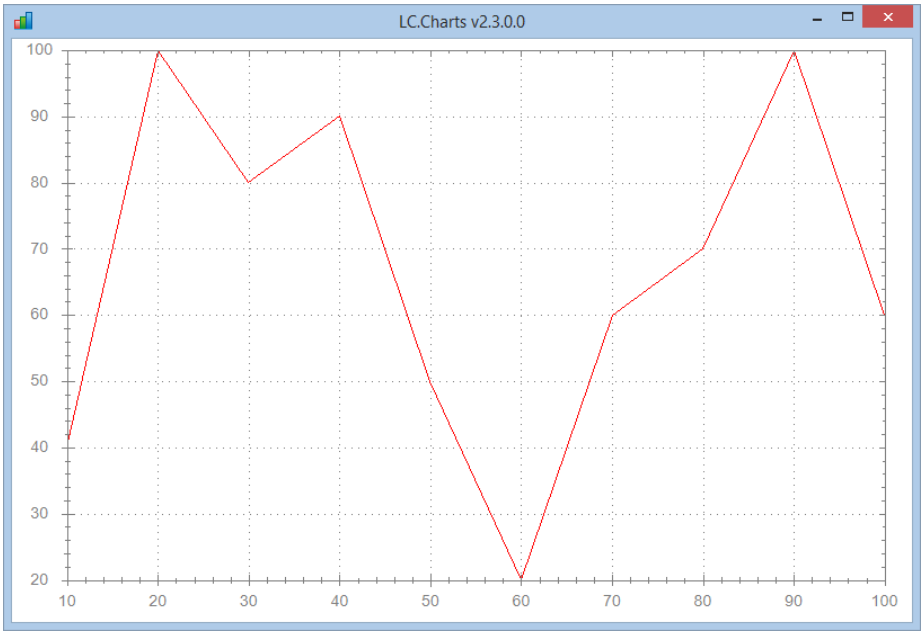

```
]chart data /nomag
```



If you use the **/nomag** option or set **xMagAuto** to **0**, you can still format the X-Axis and Y-Axis labels easily to not display all the zeroes (see the xXAxisFormat, xYAxisFormat and xY2AxisFormat properties)

Example:

```
]chart data /nomag
⎕wi'*xXAxisFormat' '0,'     ⍝ format X-labels by dividing them by 1000
⎕wi'*xYAxisFormat' '0,,'    ⍝ format Y-labels by dividing them by 1000000
⎕wi'*XRedraw'
```



Customizing the axis labels (new in v2.3.0.0)

Starting with v2.3.0.0, the auto magnitude feature is disabled by default: therefore the labels along the axis (X-Axis, Y-Axis and secondary Y-Axis) are now always in sync with the real X and Y values.

However you can easily use the new x_AxisFormat properties to format the axis labels exactly as you want (see examples further on).

You can use the following properties:

- **xXAxisFormat** to format the X-Axis labels
- **xYAxisFormat** to format the Y-Axis labels
- **xY2AxisFormat** to format the secondary Y-Axis labels

The values you provide to these properties must conform to the .Net format strings (namely the DateTimeFormatInfo and NumberFormatInfo classes possibilities).

You can find a good cheat sheet summaraizing all the .Net formatting capabilities at:

http://www.cheat-sheets.org/saved-copy/msnet-formatting-strings.pdf

For example:

'ff'□wi'*xYAxisFormat' 'f0'
would format the Y values as 0 5000 10000 15000 …

'ff'□wi'*xYAxisFormat' 'f2'
would format them as: 0.00 5000.00 10000.00 15000.00 …

'ff'□wi'*xYAxisFormat' '0,'
would format them as : 0 5 10 15 … (i.e. values divided by 1000[3])

'ff'□wi'*xYAxisFormat' '0,k'
would format them as : 0k 5k 10k 15k … (i.e. values divided by 1000[4] but with k appended)

'ff'□wi'*xXAxisFormat' '0e+0'
would format them as: 0e+0 5e+3 1e+4 2e+4 2e+4

'ff'□wi'*xXAxisFormat' '#.0e+0'
would format them as: 0.0e+0 5.0e+3 1.0e+4 1.5e+4 2.0e+4

Etc.

.Net provides extremely powerful formatting capabilities and you can basically display your Axis labels in any format you want.

[3] Using a format of '0,,' would divide values by 1000000, etc.
[4] Using a format of '0,,' would divide values by 1000000, etc.

Use a secondary Y axis (new in v2.3.0.0)

If you have several curves to draw which use very different ranges of values, you can use a secondary Y axis in your chart.

To do that, you must:

- Draw you first curve normally
- Set the **xOverlay** property to **1**
- Set the **xUseY2Axis** property to **1**
- Draw your second curve or bar chart that uses the secondary axis

You can then customize your secondary Y axis, using the following properties:

- **xY2AxisTitle**
- **xY2MajorStep**
- **xY2Max**
- **xY2MinorStep**
- **xY2Min**

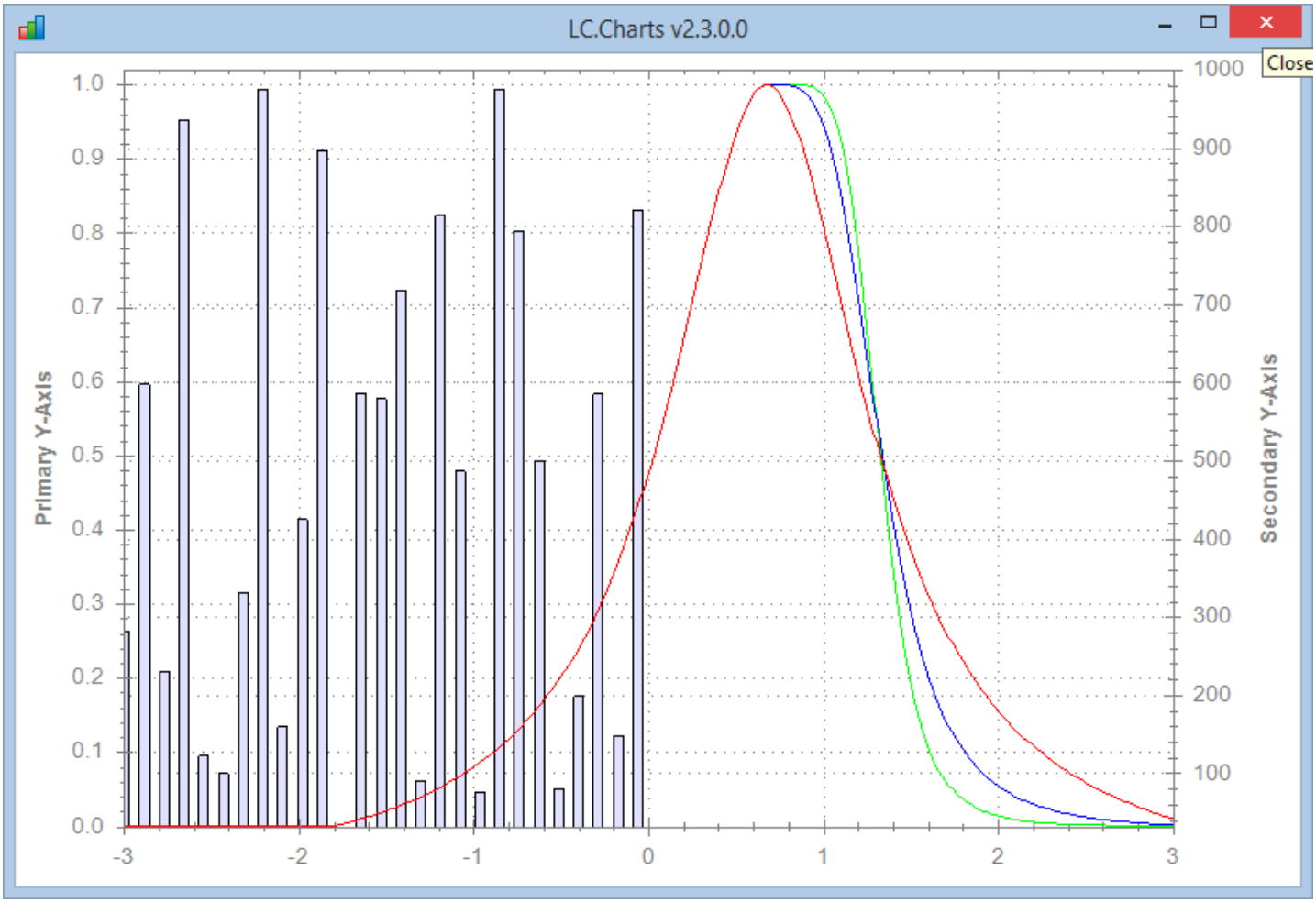
These properties work like their xYAxisTitle, xYMajorStep, xYMax, xYMinorStep and xYMin counterpart properties.

The Overlay3 APL function (new in v2.3.0.0)

The new **Overlay3** function delivered in the LCCHARTS user command file demonstrates using a chart with a secondary Y-Axis:

```
]uload Overlay3 SampleData1 SampleData2

Overlay3
```



Adding text annotations on your charts (new in v2.3.0.0)

It is now possible to add text annotations anywhere on your charts.

For that, use the new **XAddText** method, which syntax is:

```
'AddText' text x y border font
```

text should be any text possibly including embedded □tclf
x should be the abscissa of the bottom left corner of your text (in chart X-axis units)
y should be the ordinate of the bottom left corner of your text (in chart Y-axis units)
border should be defined as: (borderFlag)(borderColor)(fillColor[5])
font should be defined as: (fontname)(fontsize)(fontstyle)(fontRGBcolor)

[5] If you use only one RGB color, you'll create a gradient from white to this RGB color
You can use: **(startRGBcolor)(endRGBcolor)angle** to create a customized gradient fill color
You can use: **(startRGBcolor)(startRGBcolor)0** to create a plain solid color

Example:

```
      textBorder←1(0 0 192)(140 140 255)
      textFont←'Helvetica'12 1(0 0 192)
      'ff'⎕wi'*XAddText'('Euro was pretty strong',⎕tclf,'here a year ago!')13 200
textBorder textFont
      'ff'⎕wi'*XAddText' 'Euro is week now!'24 40 textBorder textFont
      'ff'⎕wi'*XAddText' 'The future!'32 150 textBorder textFont
```

The TopRight APL utility function and the /tr option (new in v2.3.0.0)

You can use the TopRight utility function at any time to force a window to be displayed at the top right of the screen and topmost (if it supports the xTopMost property).

Example:

```
      ]chart ?10⍴10

      TopRight ⎕wself
```

To make is easier, a **/tr** (i.e. **tr** for **T**op**R**ight) option has been added to both the **]chart** and the **]chart3d** user commands.

The above 2 expressions could therefore be condensed in the following single expression:

```
]chart ?10⍴10 /tr
```

The **TopRight** APL function and the /tr option are handy since you can both:

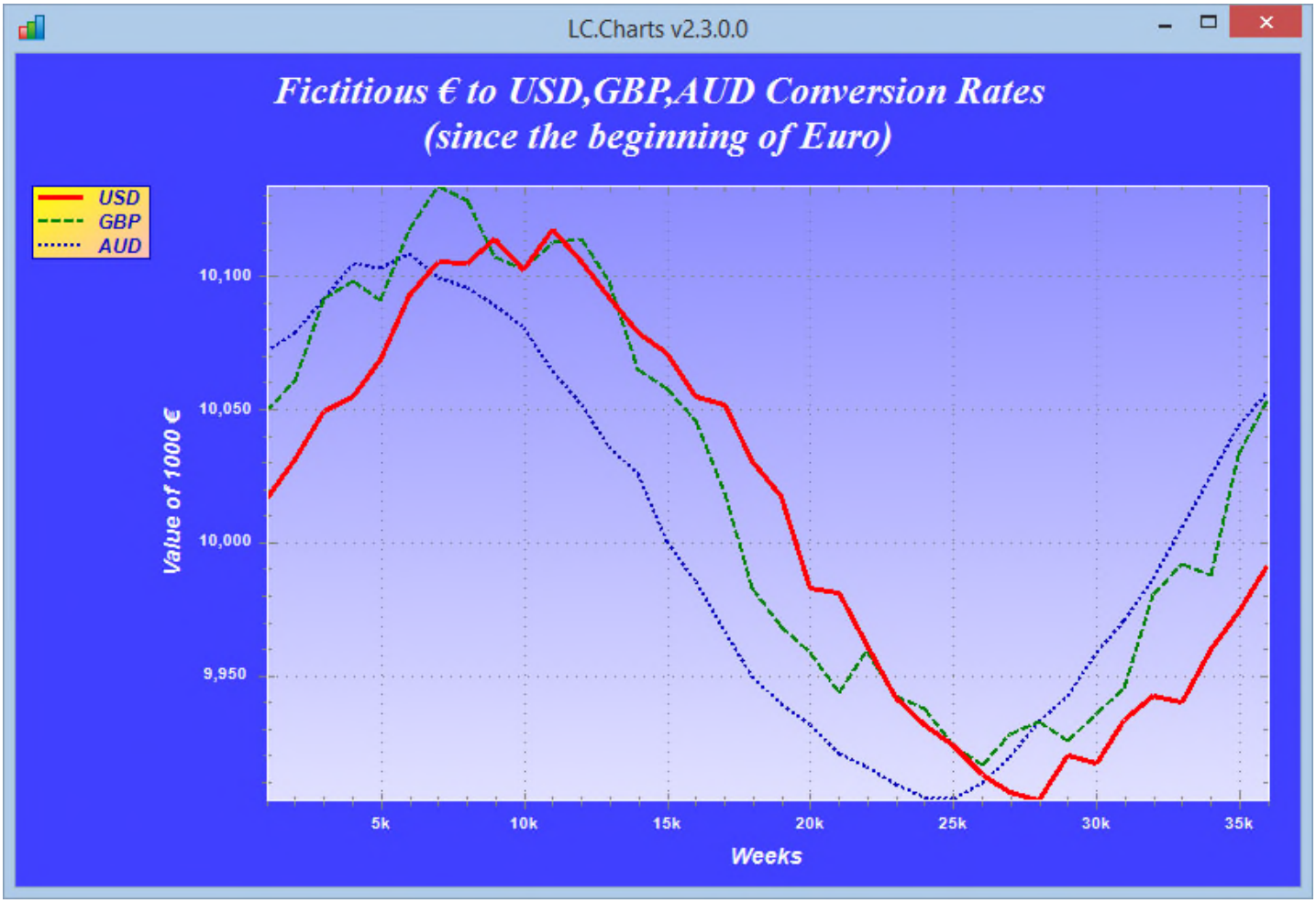
- See more of your APL Session while still seeing the form
- See the form even when you type in the APL Session

The new Demo23 APL function (new in v2.3.0.0)

The new Demo23 APL function delivered in the LCCHARTS User Command file, demonstrates how to use all of the above new properties to customize a chart.

```
]load Demo23 TopRight

Demo23
```



Zooming and Panning

LC.Charts supports multiple zooms for 2D charts and supports panning (i.e. dragging your mouse to move the chart once you have zoomed).

To zoom, simply drag your mouse to define the rectangular area you're interested in seeing zoomed.

At any time you can use the right click menu to undo a zoom operation or all zooming and panning operations.

Disable zooming (new in v2.3.0.0)

You can disable zooming by setting the **xAllowZoom** property to **0** (it is set to **1** by default)

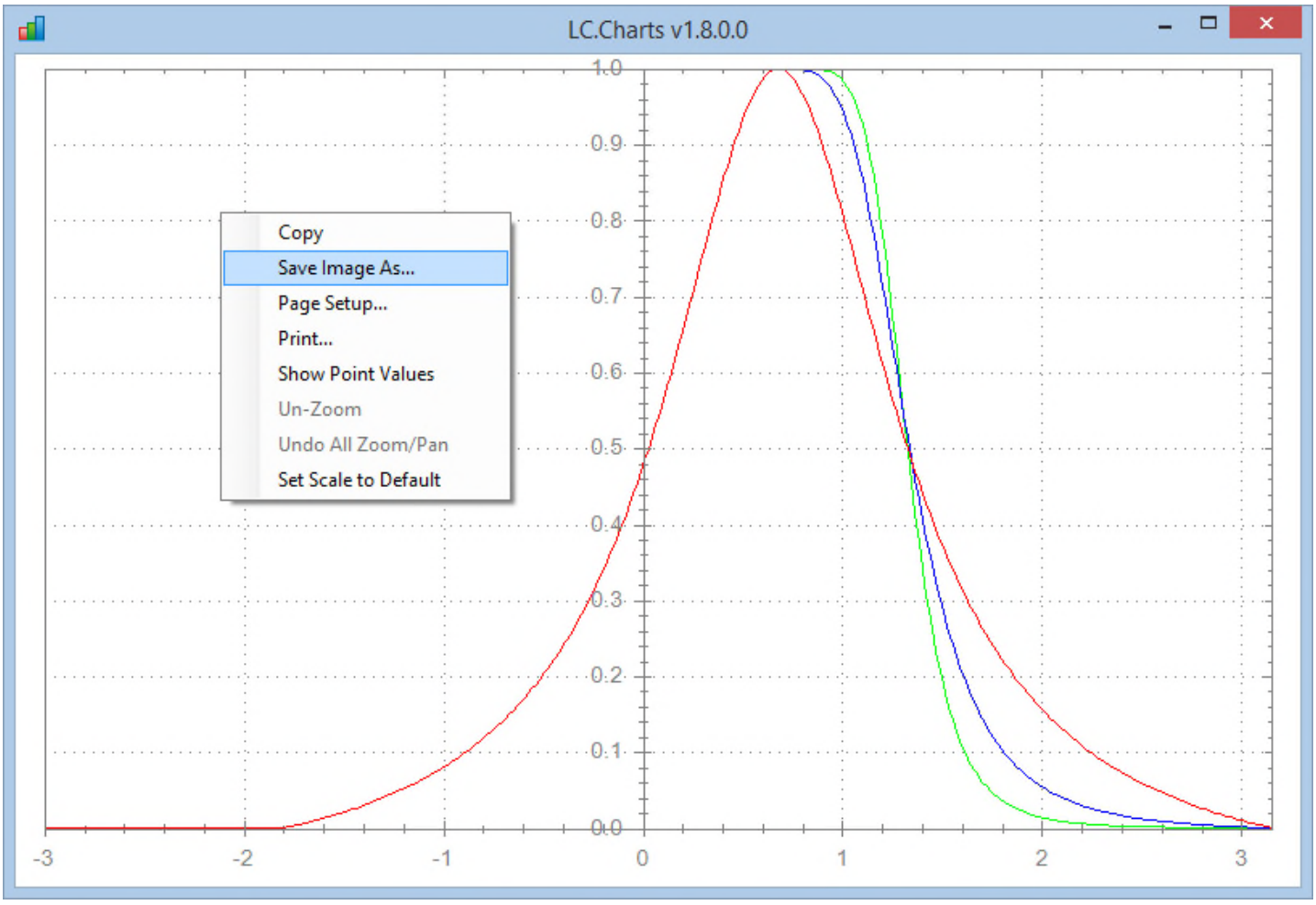
Copying, Printing, Saving chart, …

LC.Charts has a context menu which allows you to:

- Copy your chart to the Clipboard
- Save your chart to various image formats (.emf, .png, .gif, .jpg, .tif, .bmp)
- Print your chart
- Unzoom
- Undo All Zoom/Pan

Just right click anywhere on your chart at any time to use the context menu.

```
]chart SampleData1
```



Note (new in v2.3.0.0)

There is another way you can copy LC.Charts charts and print them: it is by capturing the screen.

The tool I highly recommend fr doing so I called **Snagit**[6] and is clearly the best in its category.

[6] See: http://www.techsmith.com/snagit.html

The LCCharts User Command File

The LCCharts delivered User Command file contains a few example functions and variables as well as 3 User Commands.

You can load the entire content of the LCCharts User Command file in a clear APL+Win workspace as follows[7]:

```
)clear
]uload *
```

Note that some functions need the LCCharts.sf file to be your top level UCMD file.

The Chart utility function (new in v2.1.0.0)

When working with LC.Charts you may find yourself doing the same thing over and over again, i.e. creating an LC.Charts window, setting the Reflect property to 1, making it topmost and moving it to the top right of your screen so that you could see what you type in the APL Session.

Then you can set some properties and build your chart by trial and error from the APL Session, until it is exactly as you want.

At this stage you can capture instructions from your APL+Win Session by highlighting them and pressing **Ctrl+Shift+G**[8] and pasting them into the function you're developing.

The Chart utility does exactly that.

Chart 0 ⍝ displays an LC.Charts form with an empty chart

Chart 1 ⍝ displays an LC.Charts form with some sample data

The FormChartBasic utility function (new in v2.1.0.0)

The FormChartBasic utility function is similar to the Chart utility function, but instead of displaying a C# form, it displays an APL+Win form with an LC.Charts.ChartUC control into it.

See: **Embedding Charts within APL+Win Forms** later in this document

[7] Provided LCCharts.sf is you top level User Command file.
Otherwise use:
]uload * /F={path}\LCCharts
where {path} is the full path name of the folder where you installed LCCharts.sf

[8] Ctrl+Shift+G captures only the input lines from your APL Session (much more useful than Ctrl+C)

The worlddata global variable

worlddata[9] is a 22 by 19 by 54 rank 3 array containing yearly 22 different pieces of data about 18 countries for a period of 53 years ranging from 1962 to 2014.

Here is a small excerpt of worlddata:

```
      worlddata[⍳2;⍳5;⍳5]
 Country Name           1962           1963           1964           1965
 Australia    4636880.0000000 4664710.0000000 4690090.0000000 4729760.0000000
 Austria        40410.0000000   39900.0000000   39840.0000000   39840.0000000
 Canada        699530.0000000  700810.0000000  702080.0000000  703370.0000000
 Switzerland    17334.0000000   17192.0000000   17104.0000000   17016.0000000

 Country Name           1962           1963           1964           1965
 Australia        60.3579657      60.7202270      61.0505968      61.5669786
 Austria          49.0115221      48.3929654      48.3201941      48.3201941
 Canada            7.6926291       7.7067051       7.7206711       7.7348571
 Switzerland      43.8524590      43.4932200      43.2705930      43.0479660
```

The]countries user command allows to know about the 18 countries used in worlddata and their numbers:

[9] The source for worlddata is : http://data.worldbank.org/topic/climate-change

```
      ]countries
 1 Australia          7 Finland          13 Japan
 2 Austria            8 France           14 Netherlands
 3 Canada             9 United Kingdom   15 Portugal
 4 Switzerland       10 Greece           16 Sweden
 5 Denmark           11 Ireland          17 Turkey
 6 Spain             12 Italy            18 United States
```

and the]data user command allows to know about the 22 pieces of data (planes) contained in worlddata and their respective numbers:

```
      ]data
 1 Agricultural land (km2)
 2 Agricultural land (%)
 3 Population growth (annual %)
 4 Population (total)
 5 Urban population growth (annual %)
 6 Urban population
 7 Urban population (% of total)
 8 Cereal yield (kg/ha)
 9 Population in cities > 1 million (% of total pop.)
10 Mortality rate under-5 (per 1000 live births)
11 GDP (current US$)
12 Electricity prod. coal based (% of total)
13 Electricity prod. hydroelectric based (% of total)
14 Electricity prod. natural gas based (% of total)
15 Electricity prod. nuclear based (% of total)
16 Electricity prod. oil based (% of total)
17 Electricity prod. (kWh)
18 Electricity prod. renewable sources (kWh)
19 Electricity prod. renewable sources, not hydro (kWh)
20 Electricity prod. renewable sources, not hydro (% of total)
21 Energy use (kt of oil equivalent)
22 Energy use (kg of oil equivalent per capita)
```

The WorldDataEvolution function

This function as well as the WorldDataAnimation function, together with the worlddata global variable are provided to have some real data to test LC.Charts with.

WorldDataEvolution extracts one given piece of data from worlddata from one or more countries and returns an array suitable for being used as an argument to LC.Charts.

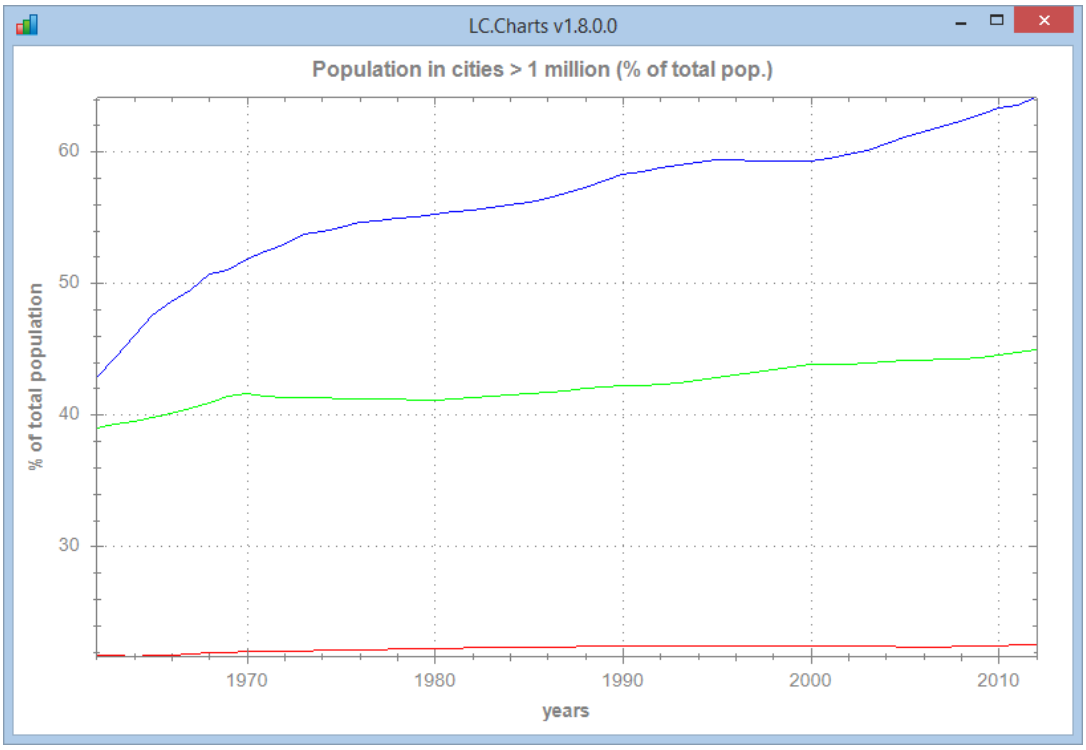
Example:

```
      ⍝ The following command extracts the evolution over years of the percentage
of total population which lives in cities over 1 million habitants

      9 WorldDataEvolution 8 18 13
 Population in cities > 1 million (% of total pop.)    France     Japan United States
                                                1962 21.738609 42.822435     39.013264
                                                1963 21.681171 44.340251     39.224184
                                                1964 21.648342 45.908228     39.461981
                                                1965 21.654162 47.515171     39.759195
                                                1966 21.706754 48.601969     40.103057
                                                1967 21.799416 49.479509     40.481856
                                                1968 21.895393 50.679835     40.907751
                                                   ⋯
```

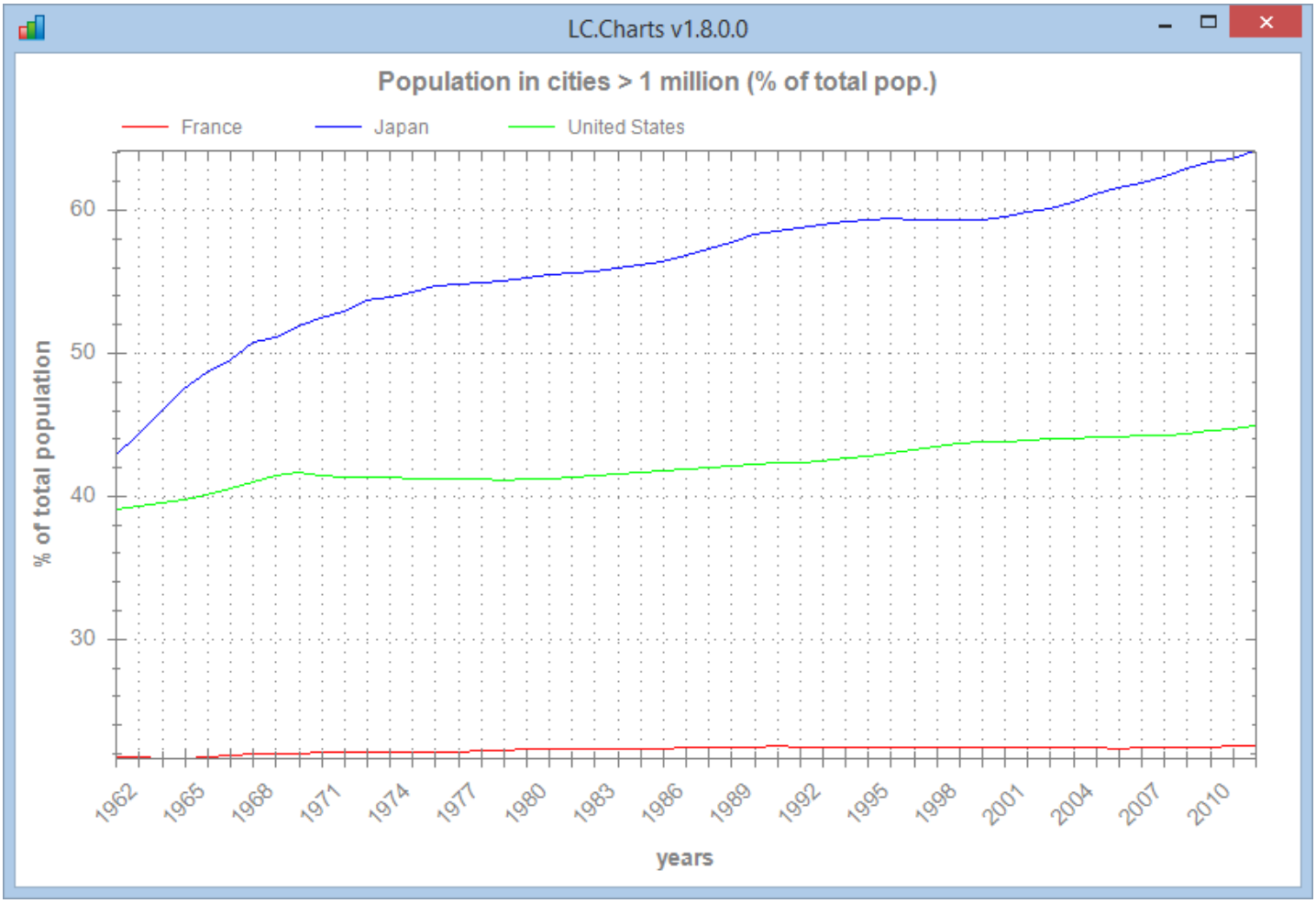
Let's display this chart:

```
      data←9 WorldDataEvolution 8 18 13
      legend←data[1;2 3 4]
      ]chart 1 0↓data /le=⍪legend /ti=⍪⊂enlist data[1;1] /xti=years /yti=% of
total population
```



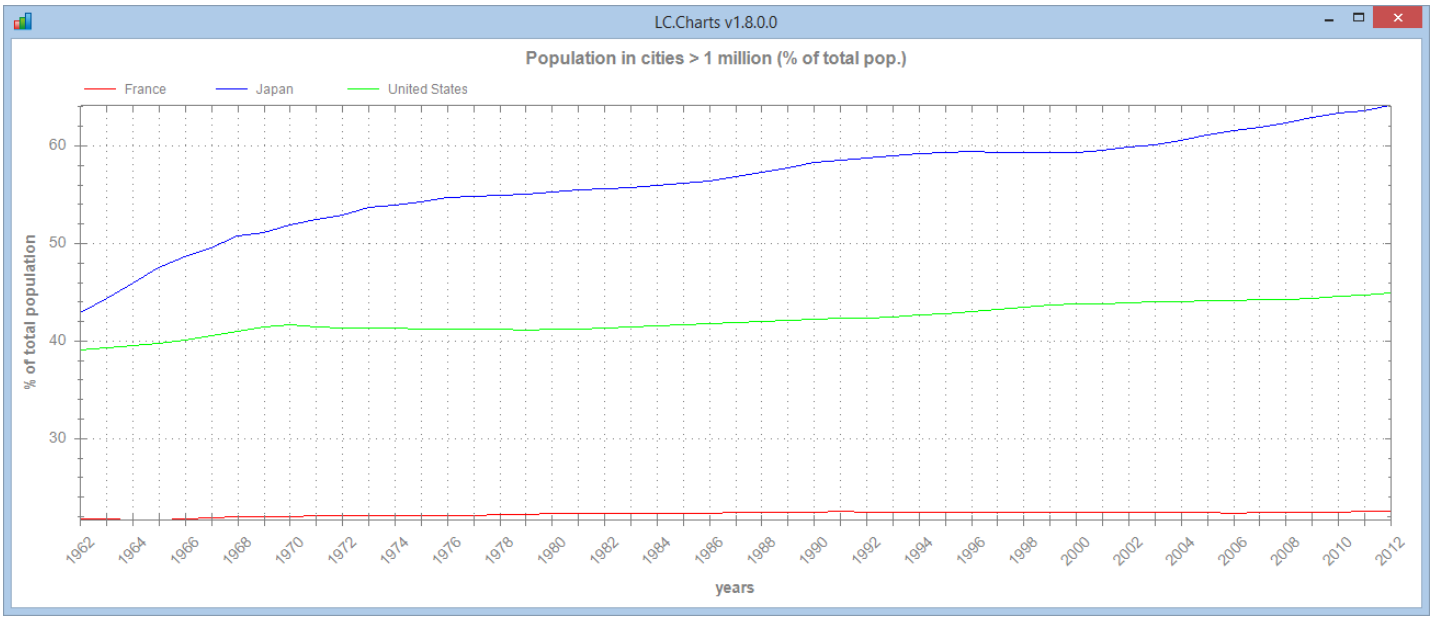
It would be nice to be able to display more of the 53 years below the X-Axis: for that, we need to use an angle and to set the XMajorStep and XMinorStep property to 1 as well:

```
      ]chart 1 0↓data /le=±legend /ti=±□enlist data[1;1] /xti=years /yti=% of
total population /xan=45 /xmajstep=1 /xminstep=1
```

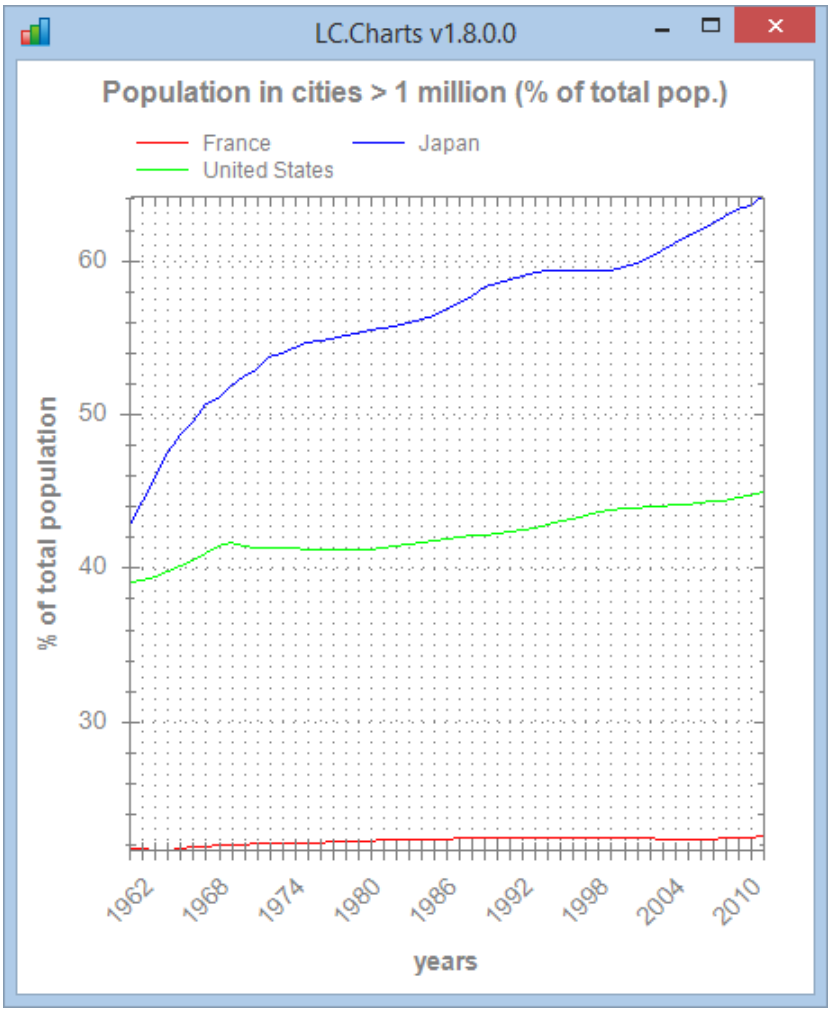


Note that LC.Charts did not display every year along the X-Axis, just because it did not have enough room.

But if we enlarge the chart, it automatically adapts the X-Axis labels:

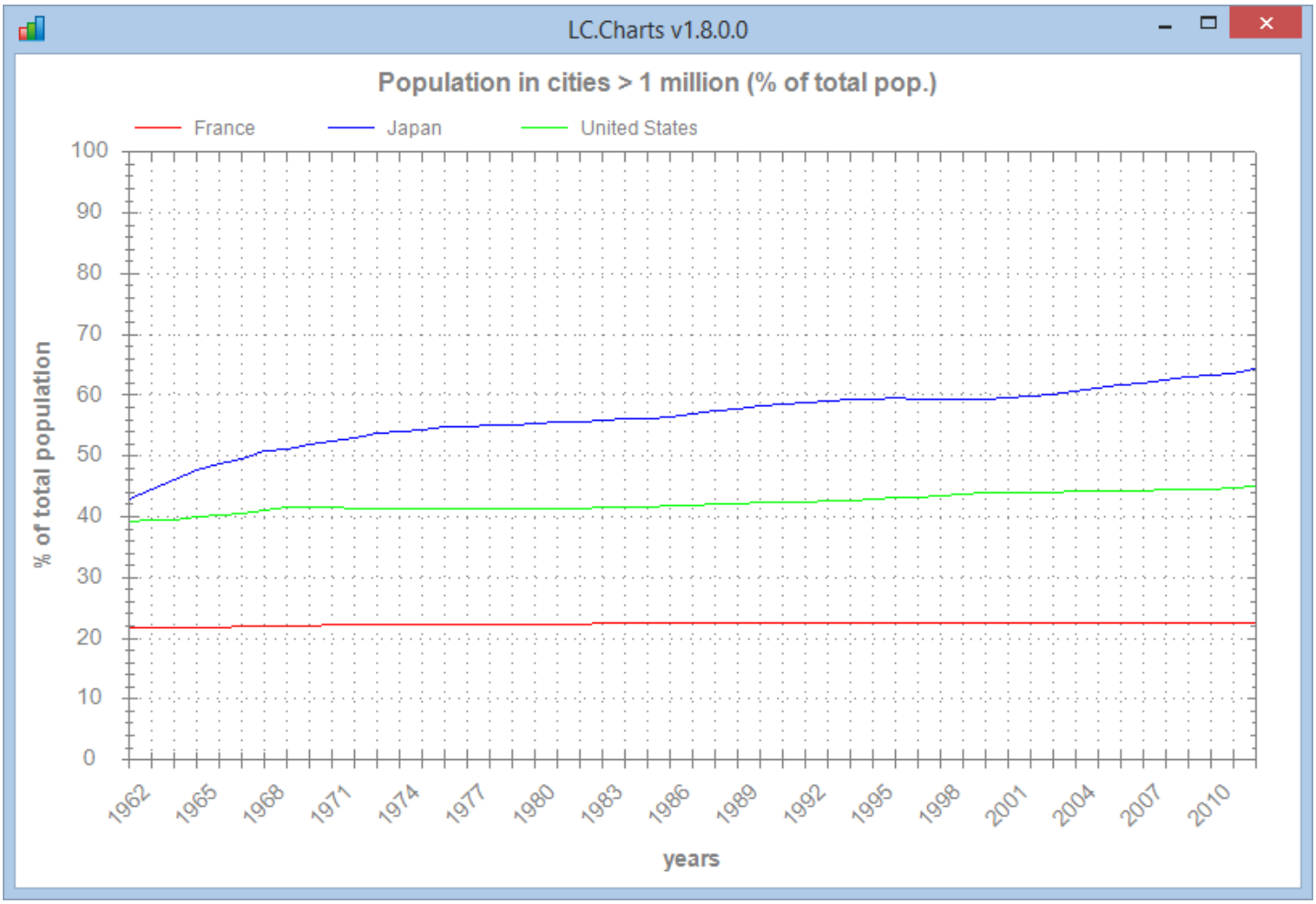


And if we make the chart narrower:



We can also force the Y-axis minimum value to 0 and maximum value to 100 (%):

```
      ]chart 1 0↓data /le=±legend /ti=±□enlist data[1;1] /xti=years /yti=% of
total population /xan=45 /xmajstep=1 /xminstep=1 /ymin=0 /ymax=100
```



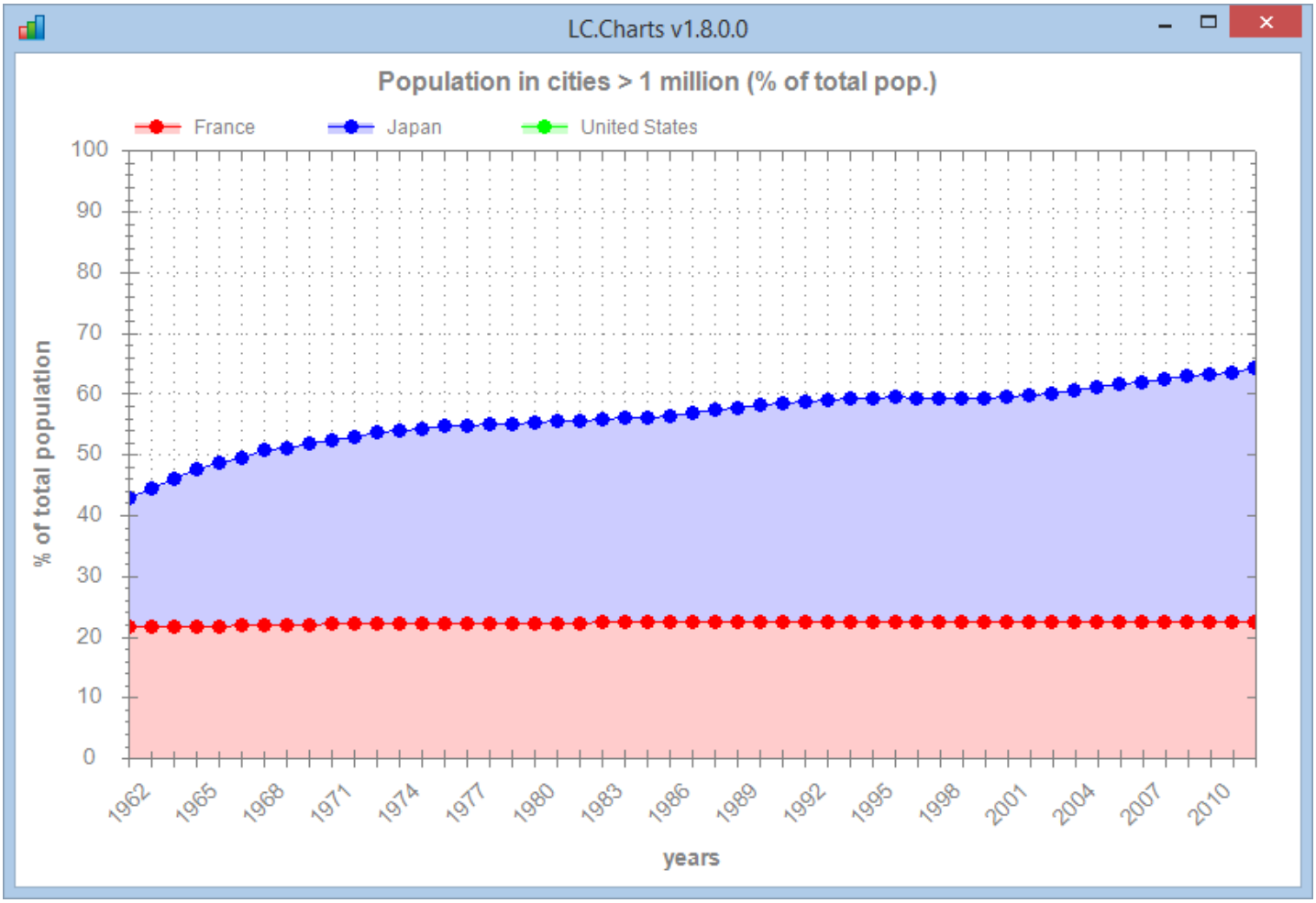
We clearly see that:

- The % of population living in large cities has been very stable in France
- The % of population living in large cities has increased but relatively moderately in the USA
- The % of population living in large cities has dramatically increased in Japan

All this could have also been achieved using a □wi approach, but would have required much more typing and at least 12 lines of code.

You can customize the chart more. We can for example add point symbols and paint the surfaces below the curves:

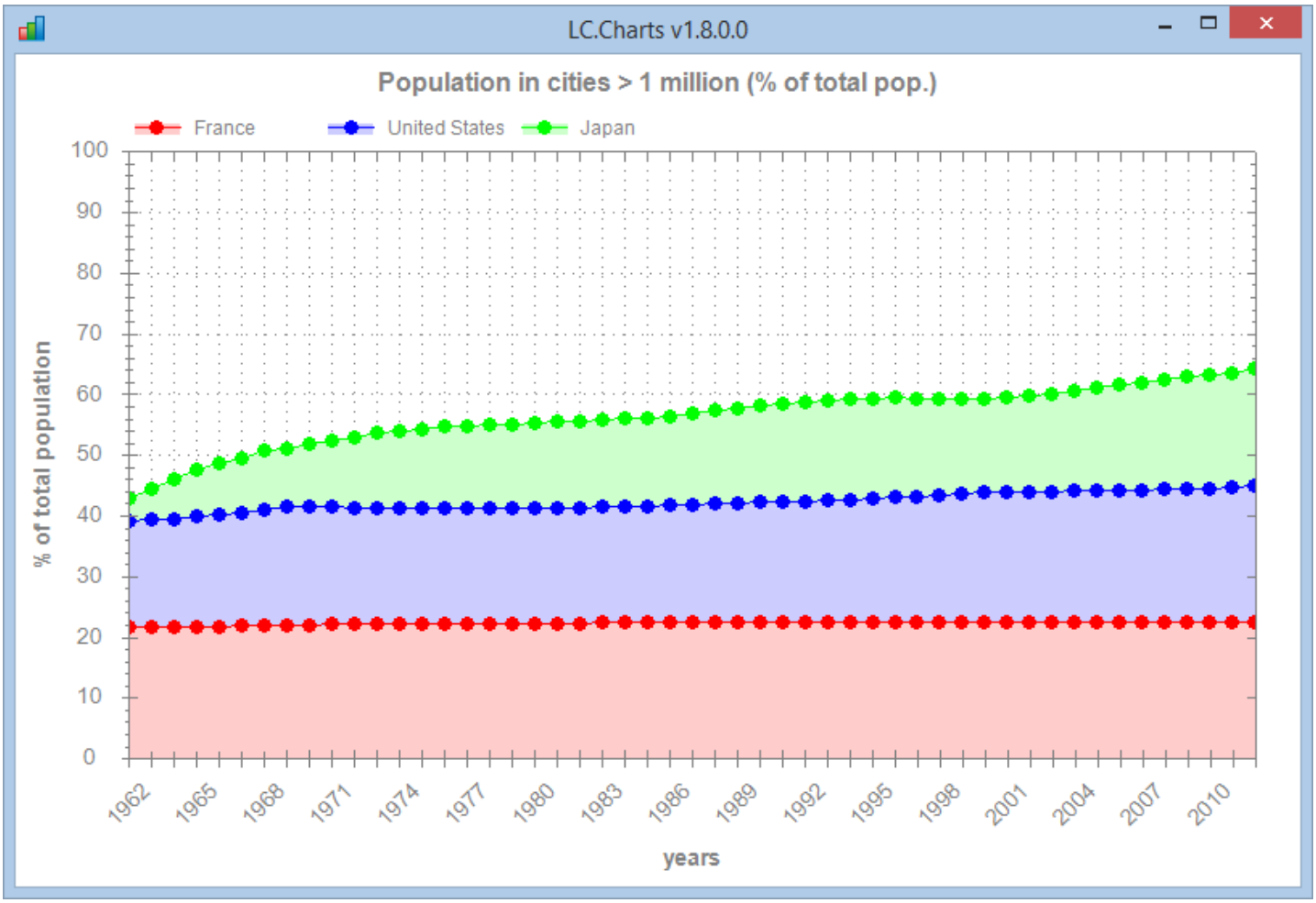
```
      ]chart 1 0↓data /le=⍷legend /ti=⍷⍋enlist data[1;1] /xti=years /yti=% of
total population /xan=45 /xmajstep=1 /xminstep=1 /ymin=0 /ymax=100 /sy=circle
/cf=80
```



Oops! The problem here is that the USA curve which is drawn after the Japan curve may not be seen because it is behind the Japan surface.

We can solve the problem as follows:

```
      data←data[;1 2 4 3]
      legend←data[1;2 3 4]
      ]chart 1 0↓data /le=⍷legend /ti=⍷⍋enlist data[1;1] /xti=years /yti=% of
total population /xan=45 /xmajstep=1 /xminstep=1 /ymin=0 /ymax=100 /sy=circle
/cf=80
```



This little tutorial has shown how you can progressively build your]chart command by adding more and more options to achieve what you want and each time just pressing Enter to check the results on your chart.

Animations

The LC.Charts.dll is so fast that it can draw a 100000 (on hundred thousand) scatter plot in less than half a second.

It can therefore be used to do very nice and fluid graphics animations.

The WorldDataAnimation function can be used for that purpose.

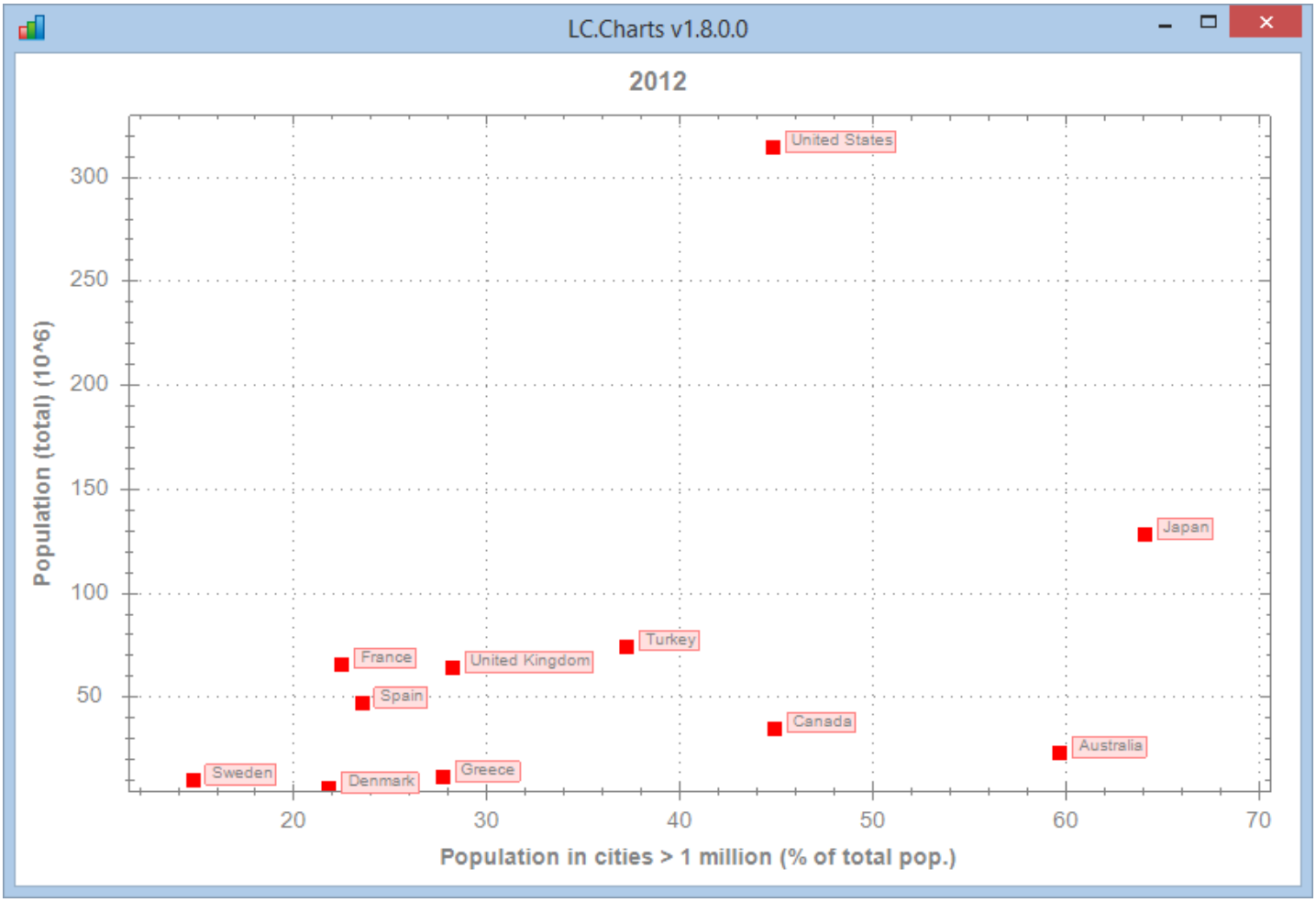
It allows you to draw one of the 22 world data mentioned above against another one for a few countries and see the evolution from 1962 to 2012 in an animated chart.

For example, try:

```
      9 4 WorldDataAnimation ''
```

9 and 4 are the number of the 2 pieces of data you want on your X and Y axis.

Here is what you get at the end of the animation:



The WorldDataAnimation function is a good example of using wi to draw a chart. It is reproduced on the next page.

You can try to cross other variables or change the delay used on line 35 of WorldDataAnimation.

```
    ∇ items WorldDataAnimation
countries;allcountries;bool;dataItems;data1;data2;i;x;y;z;⎕io
[1]   ⍝∇ items WorldDataAnimation countries -- Shows an animation of world cross
data evolution from 1961 to 2012
[2]   ⍝∇ items ↔ an integer scalar (see ]data to see possible choices)
[3]   ⍝∇ countries ↔ '' for a selection of 11 countries
[4]   ⍝∇  or a vector of country numbers (see ]countries to see possible choices)
[5]   ⍝∇ Requires: (F) Countries DataItems
[6]   ⍝∇           (V) worlddata (source: data downloaded from
http://data.worldbank.org/topic/climate-change)
[7]   ⍝∇ (c) 2015 Lescasse Consulting
[8]   ⍝∇ ELE6jan15
[9]
[10]  ⎕io←1
[11]  dataItems←DataItems[items]
[12]  allcountries←Countries
[13]  :if 0∊⍴countries ⋄ countries←1 3 5 6 8 9 10 13 16 17 18 ⋄ :endif
[14]  bool←1,(⍳⍴allcountries)∊countries
[15]  (data1 data2)←⊂[2 3]bool/[2]worlddata[items;;]
[16]
[17]  z←'ff'⎕wi'*Create' 'LC.Charts.Chart'
[18]  z←'ff'⎕wi'*XShow'
[19]  z←'ff'⎕wi'*xXMin'(⌊/,1 0↓0 1↓0 ¯2↓data1)
[20]  z←'ff'⎕wi'*xXMax'(1.1×⌈/,1 0↓0 1↓0 ¯2↓data1)
[21]  z←'ff'⎕wi'*xYMin'(⌊/,1 0↓0 1↓0 ¯2↓data2)
[22]  z←'ff'⎕wi'*xYMax'(1.05×⌈/,1 0↓0 1↓0 ¯2↓data2)
[23]  z←'ff'⎕wi'*xSymbol' 'square'
[24]  z←'ff'⎕wi'*xSymbolSize'8
[25]  z←'ff'⎕wi'*xType' 'scatter'
[26]  z←'ff'⎕wi'*xXAxisTitle'(1⊃dataItems)
[27]  z←'ff'⎕wi'*xYAxisTitle'(2⊃dataItems)
[28]  :for i :in 1↓¯2↓⍳2⊃⍴data1
[29]      x←1↓data1[;i]
[30]      y←1↓data2[;i]
[31]      z←'ff'⎕wi'*xData'(x,[1.5]y)(1↓data1[;1])
[32]      z←'ff'⎕wi'*xPointLabels'1
[33]      z←'ff'⎕wi'*xTitle'(⍕data1[1;i])
[34]      z←'ff'⎕wi'*XRedraw'
[35]      z←⎕dl.1
[36]  :endfor
    ∇
```

The Lissajous Curve

Another nice data animation is the Lissajous[10] mathematical curve.

Try the following[11]:

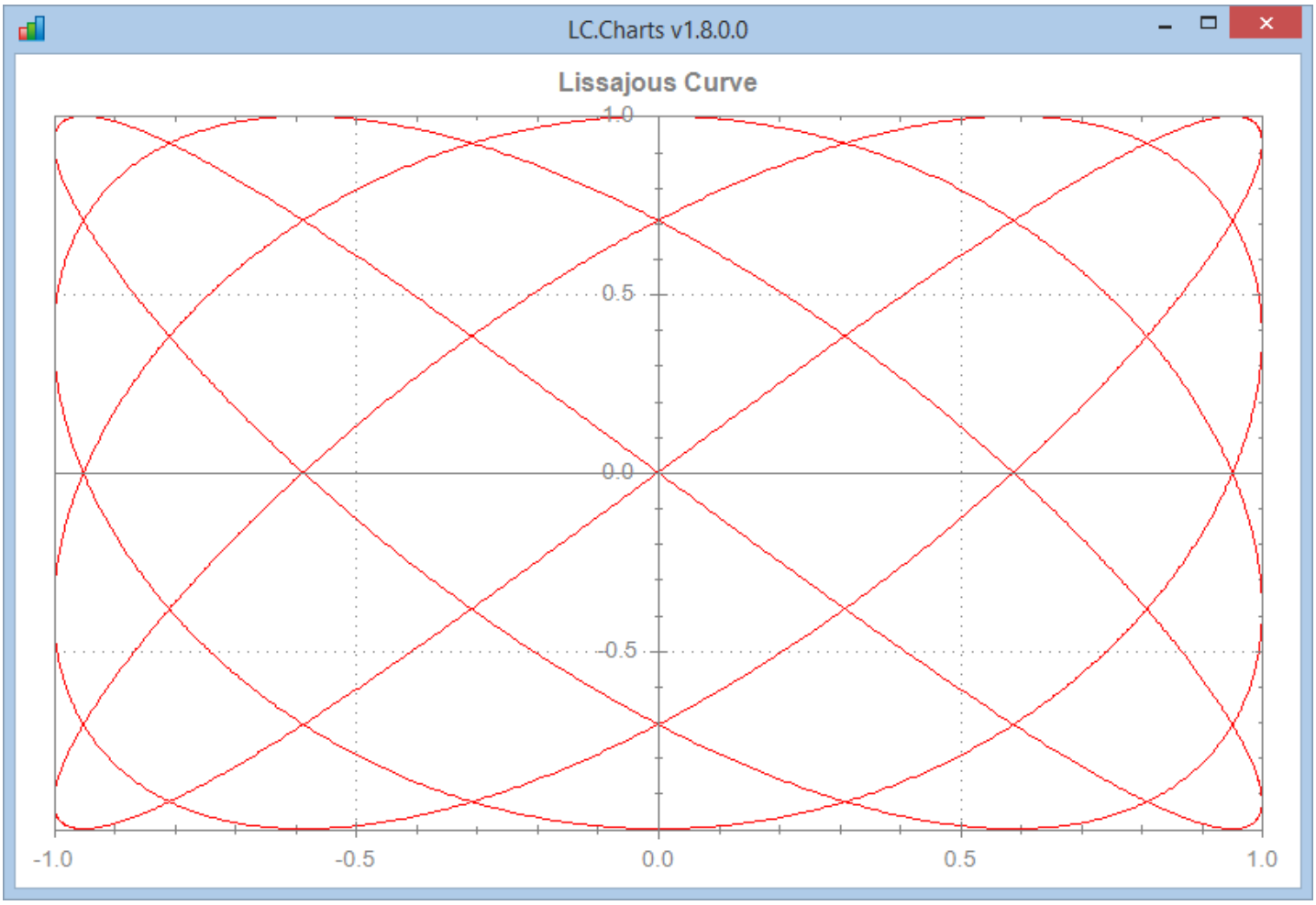
```
Lissajous 1 2

Lissajous 3 4

Lissajous 19 20

Lissajous 39 40
```

Or try any 2 integer values you'd like!



[10] Jules Antoine Lissajous was a French Mathematician (1822-1880). The Lissajous curve is a mathematical figure showing a type of harmonic motion [Wikipedia]

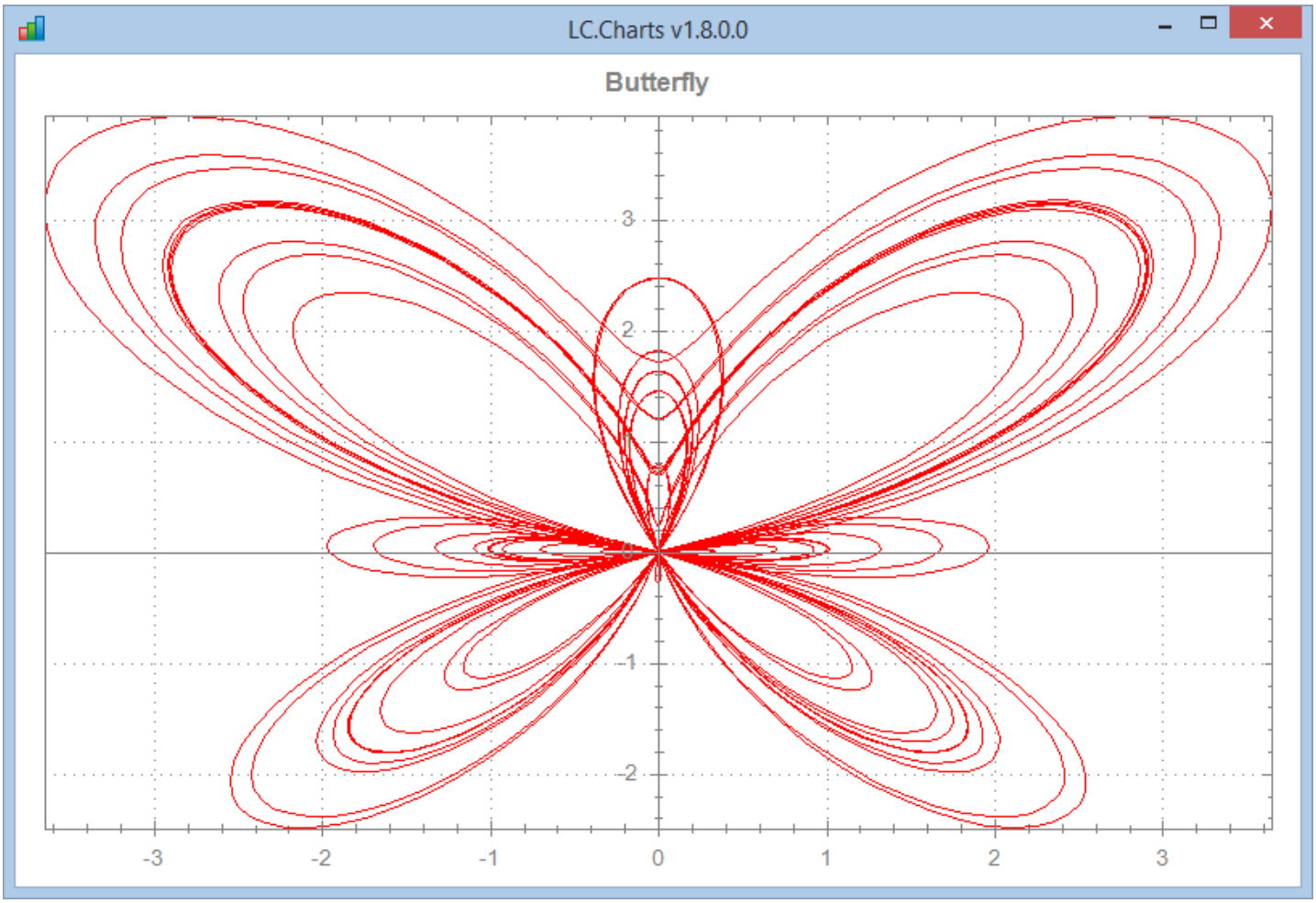
[11] The LCCharts.sf file must be your top level UCMD file for the Lissajous function to work !

The Butterfly Curve

Do you believe that a simple mathematical cartesian equation can draw a sophisticated butterfly?

If you're not convinced, run the Butterfly function and look at its code:

```
Butterfly
```



It's a curiosity, but also shows how fast LC.Charts is.

Combining Charts: the Overlay1 and Overlay2 functions (new in v2.1.0.0)

LC.Charts v2.1.0.0 introduces a new property called Overlay and a new]chart option called /overlay.

This allows you to combine several graphics in the same chart: for example, you can display a bar chart and on top of it a set of curves, etc.

First, when combining charts, always remember that the 1st chart you draw will be the topmost one, then the 2nd chart will get displayed behind the 1st, the 3rd will be displayed behind the 2nd, etc.

To efficiently combine charts proceed as follows:

1. Be sure to set the xXMin, xXMax, xYMin and xYMax properties in the 1st chart you create
2. Do not change these properties in the other charts you add to the 1st chart
3. Display your 1st chart
4. Then set the xOverlay property to 1 ot use the /overlay option on the 2nd]chart command you use
5. Display your 2nd chart on the same LC.Charts instance

Also, it is advisable to change de default LC.Charts colors (**xChartColors** property) to light colors for the 2nd, 3rd, etc. charts so that the first chart be well visible.

Here is some sample code doing a combined chart using]chart:

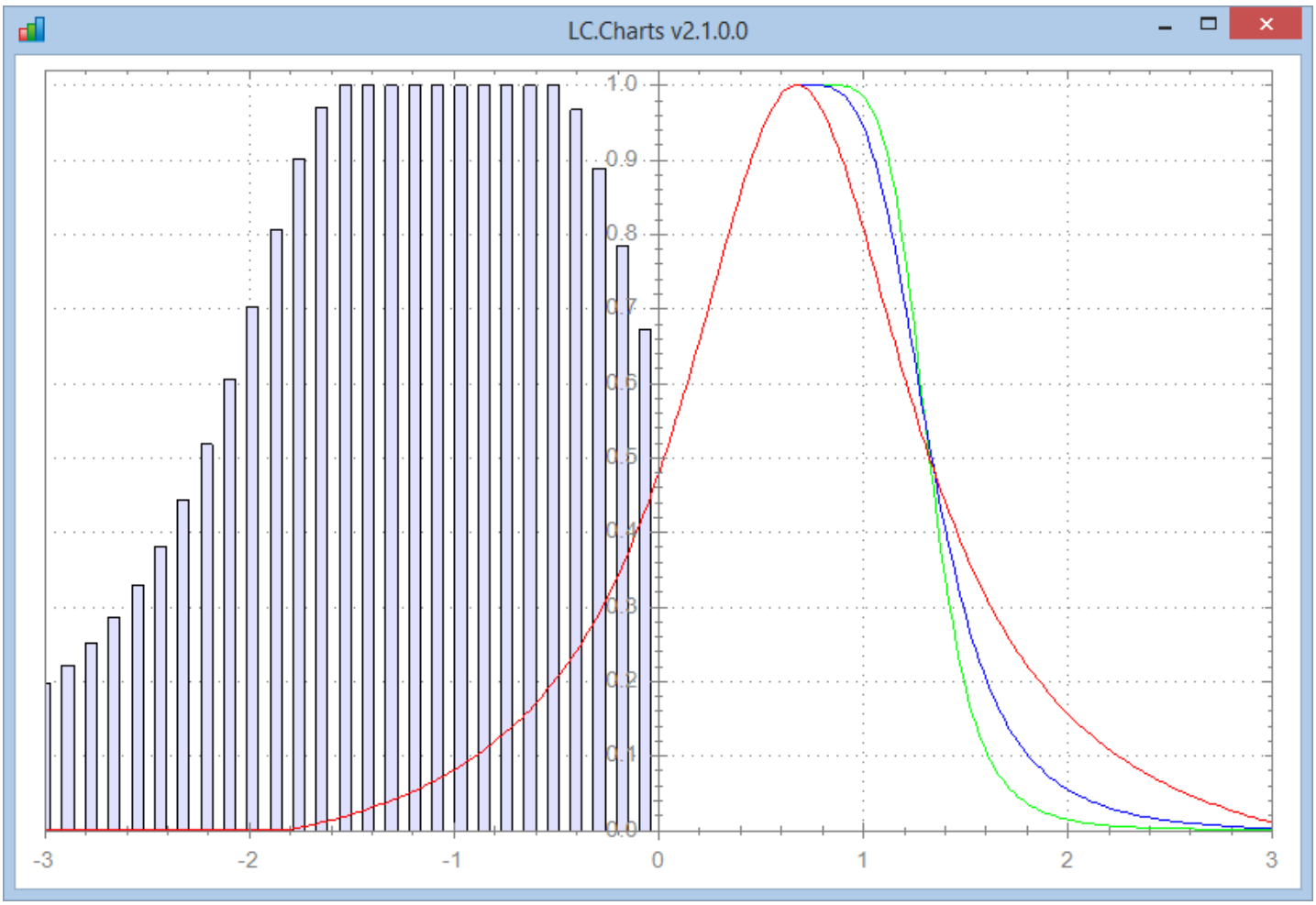
```
]uload SampleData1 SampleData2
]chart SampleData1 /xmin=-3 /xmax=3 /ymin=0 /ymax=1.02 /to
colors1←(⊂224 224 255),?21⍴⊂3⍴255
]chart (160⍴6↑1)≠SampleData2[;1 2] /chartcolors=⊂colors1 /ty=bar /overlay
```

And the same combined chart using □wi:

```
'ff'□wi'*Create' 'LC.Charts.Chart'('*xTopMost'1)
'ff'□wi'*Set'('*xXMin'¯3)('*xXMax'3)('*xYMin'0)('*xYMax'1.02)
'ff'□wi'*xData'SampleData1
colors1←(⊂224 224 255),?21⍴⊂3⍴255
'ff'□wi'*xOverlay'1
'ff'□wi'*xChartColors'colors1
'ff'□wi'*xType' 'bar'
'ff'□wi'*xData'((160⍴6↑1)≠SampleData2[;1 2])
'ff'□wi'*XShow'
```

The new **Overlay1** and **Overlay2** APL functions in the LCCHARTS UCMD file show examples of combined charts.

```
Overlay2
```



Embedding Charts within APL+Win Forms (new in v2.1.0.0)

Embedding Charts Basics

Stating with version 2.1.0.0, it is now possible to embed charts built with LC.Charts within APL+Win Forms.

This is done through the use of a new object called **LC.Charts.ChartUC** (UC for User Control).

In order for embedding a chart in an APL+Win Form, you must:

1. Create an instance of your APL+Win Form
2. Create an instance of LC.Charts.ChartUC as a child of your APL+Win Form
3. Call the SetParent Windows 32 API to force the LC.Charts.ChartUC instance object to become a real child of the APL+Win Form

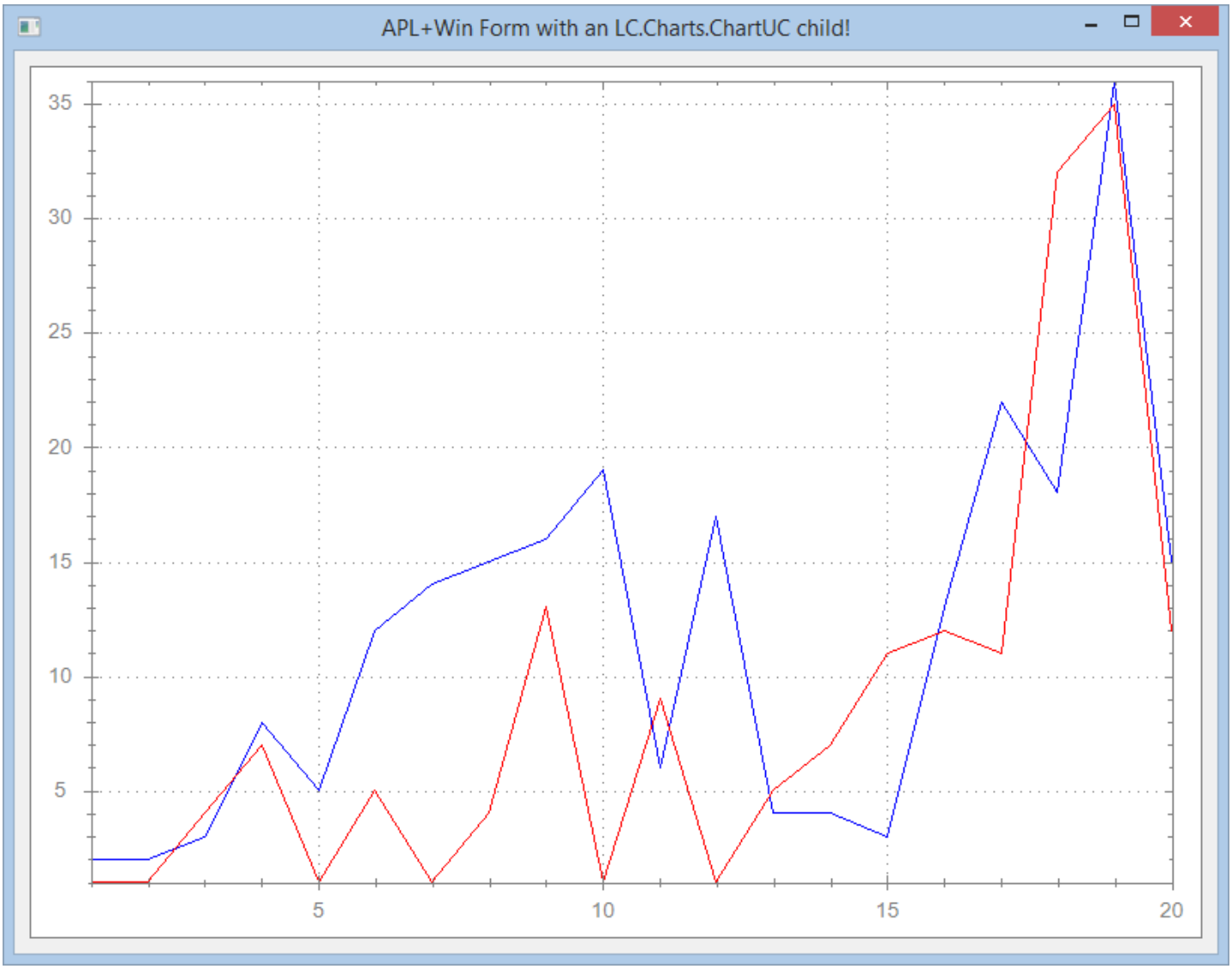
This is indeed pretty simple and is best demonstrated by the following **FormChartBasic** APL function (delivered in the LCCHARTS.SF User Commands file):

```
      ]uload FormChartBasic

    ∇ FormChartBasic a;c;d;data;z;⎕wself
[1]   ⍝∇ FormChartBasic a -- Creates an APL Form with an LC.Charts.ChartUC control
[2]   ⍝∇ (c) 2015 Lescasse Consulting
[3]   ⍝∇ ELE15jan15
[4]
[5]   :select a
[6]   :case''
[7]       z←'ff'⎕wi'*Create' 'Form'('*scale'5)('*size'600 800)'*Hide'
[8]       z←'ff'⎕wi'*caption' 'APL+Win Form with an LC.Charts.ChartUC child!'
[9]       z←'ff'⎕wi'*.cc.Create' 'LC.Charts.ChartUC'('*xWhere'10 10 580 780)
[10]      z←⎕wcall'SetParent'('ff'⎕wi'*.cc.xHandle')('ff'⎕wi'*hwnd')
[11]      z←'ff'⎕wi'*.cc.xData'(data←(⍳20),?20 2⍴⍳100000)
[12]      z←'ff'⎕wi'*.cc.xSymbol' 'square'
[13]      z←'ff'⎕wi'*.cc.XShow'
[14]      z←'ff'⎕wi'*Show'
[15]      ⍝ Events
[16]      z←'ff'⎕wi'*onResize' 'FormChartBasic"onResize"'
[17]  :case'onResize'
[18]      (c d)←⎕wi'*size'
[19]      z←⎕wi'*.cc.xWhere'10 10,c d-20
[20]  :endselect
    ∇
```

Let's run this function:

```
FormChartBasic''
```



Try resizing the APL Form: as you see the chart resizes nicely when you resize the form.

This is quite interesting as you may therefore include LC.Charts charts in APL+Win forms that contain any number of APL+Win controls.

Differences between LC.Charts.UC and LC.Charts

In general, LC.Charts.UC and LC.Charts have the same properties and methods, and the same documentation, so if you know how to use LC.Charts, you'll know how to use LC.Charts.UC.

However there are a few differences which are important to note:

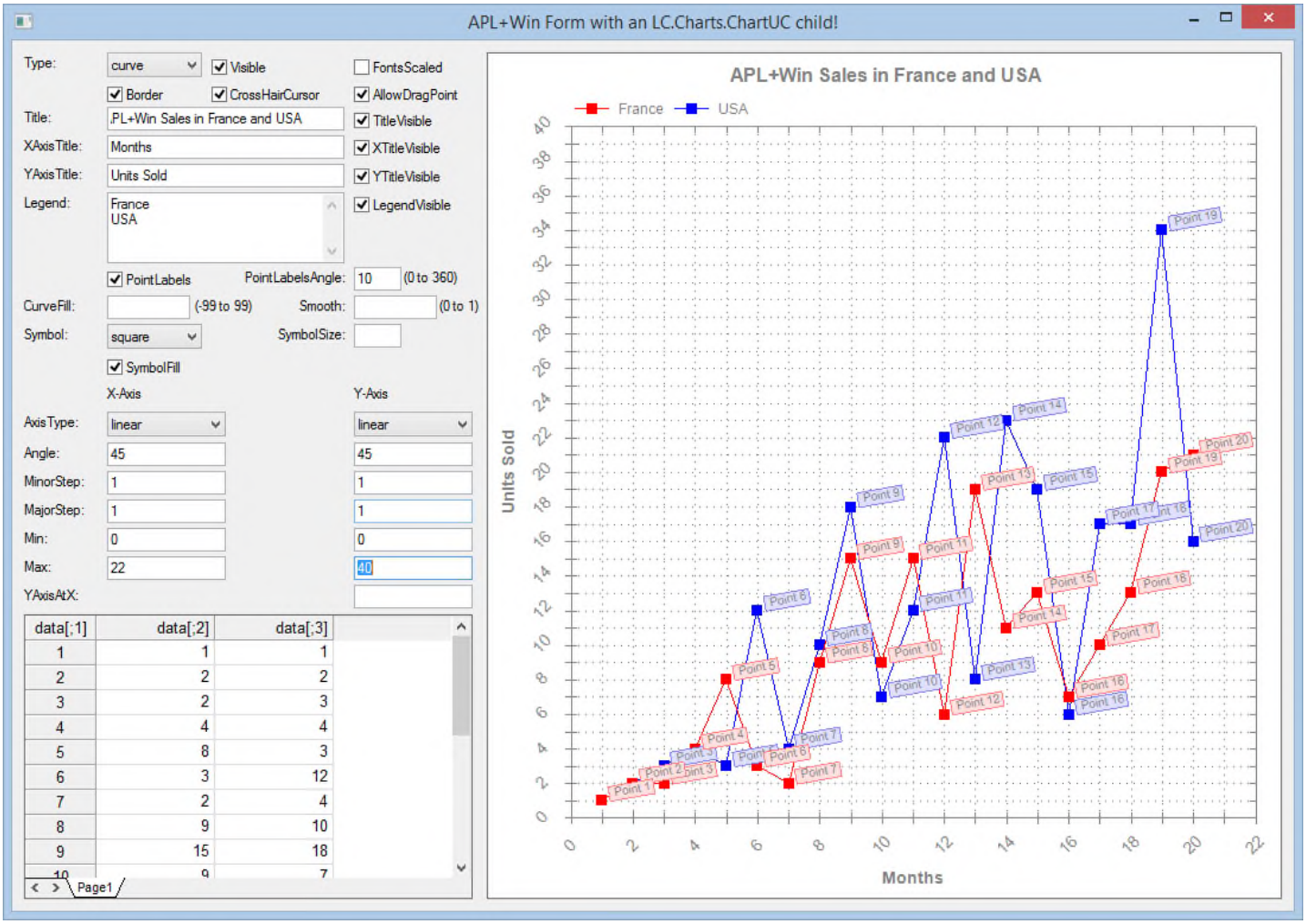
1. LC.Charts.ChartUC may only be used with □wi (i.e. the]chart command does not apply to LC.Charts.ChartUC)
2. LC.Charts.ChartUC has an **xBorder** property (set to **1** by default)
3. LC.Charts.ChartUC has an **xVisible** property (set to **1** by default)
4. LC.Charts.ChartUC does not have an **xCaption** property
5. LC.Charts.ChartUC does not have an **xTopMost** property
6. LC.Charts.ChartUC does not have an **XClose** method
7. LC.Charts.ChartUC does not have an **XWait** method

The LC.Charts Tester and Function Generator (new in v2.1.0.0)

Version 2.1.0.0 also includes an APL function called FormChart which is an LC.Charts Tester and Function Generator.

Just run:

```
FormChart''
```



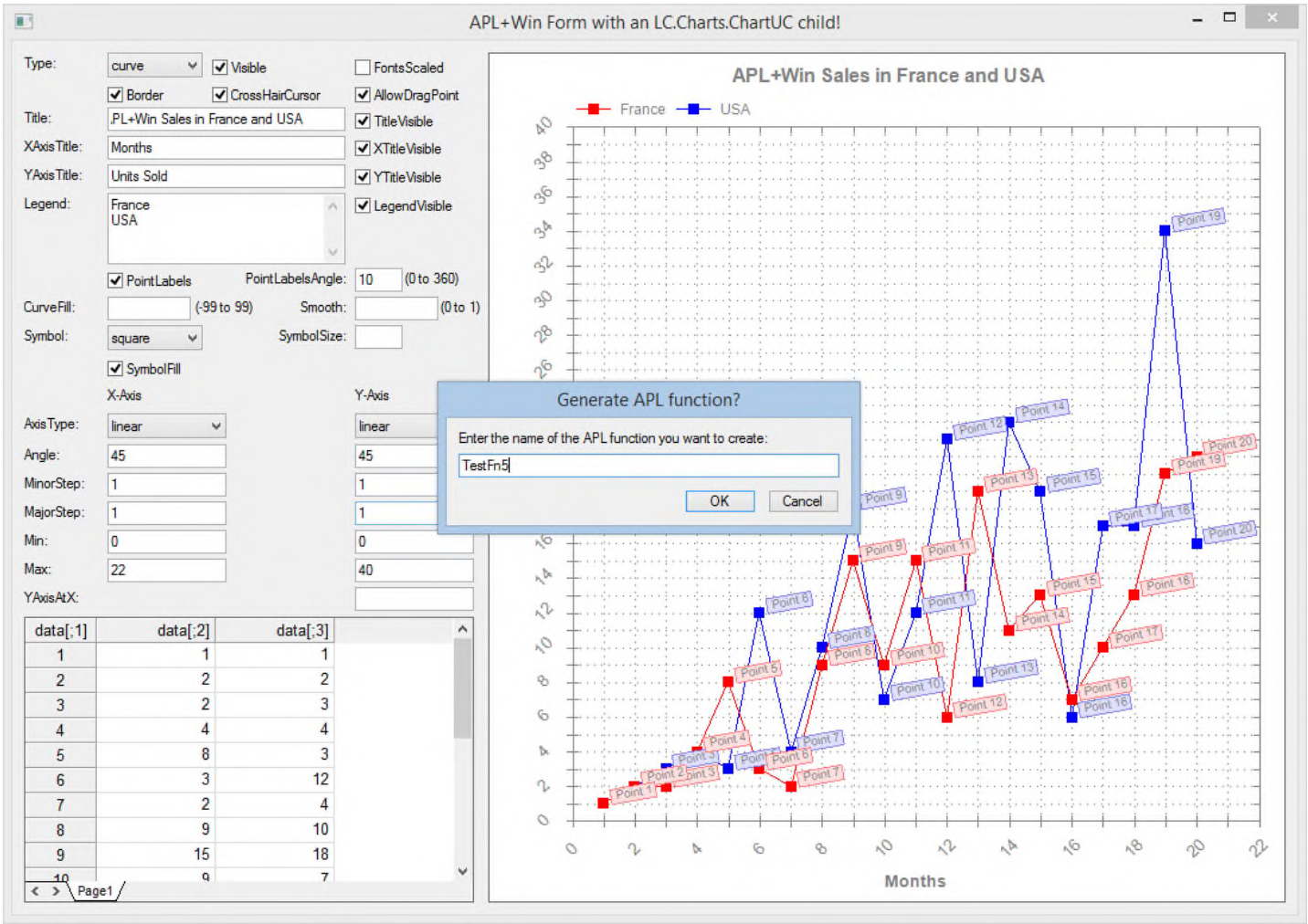
This Form's layout has been generated with zObjects (read: http://www.lescasse.com/downloads/zObjects.pdf)

Just enter the desired values in the fields and see the chart auto adjusting itself according to your choices.

Note that some fields subscribe to the onChange event, i.e. the chart reacts as you type in the field. Some other fields subscrive to the onExit event, i.e. the chart reacts only when you exit the field (this is in particular the case of most of the X-Axis and Y-Axis fields).

Once you are done visually customizing the chart to your best wishes, close the FormChart form.

As you do so, you are prompted to enter the name of a new APL function:



You must enter a name having more than one character and you may not enter the name of an APL function in the workspace.

Then click OK and a new function will have been generated in your APL workspace.

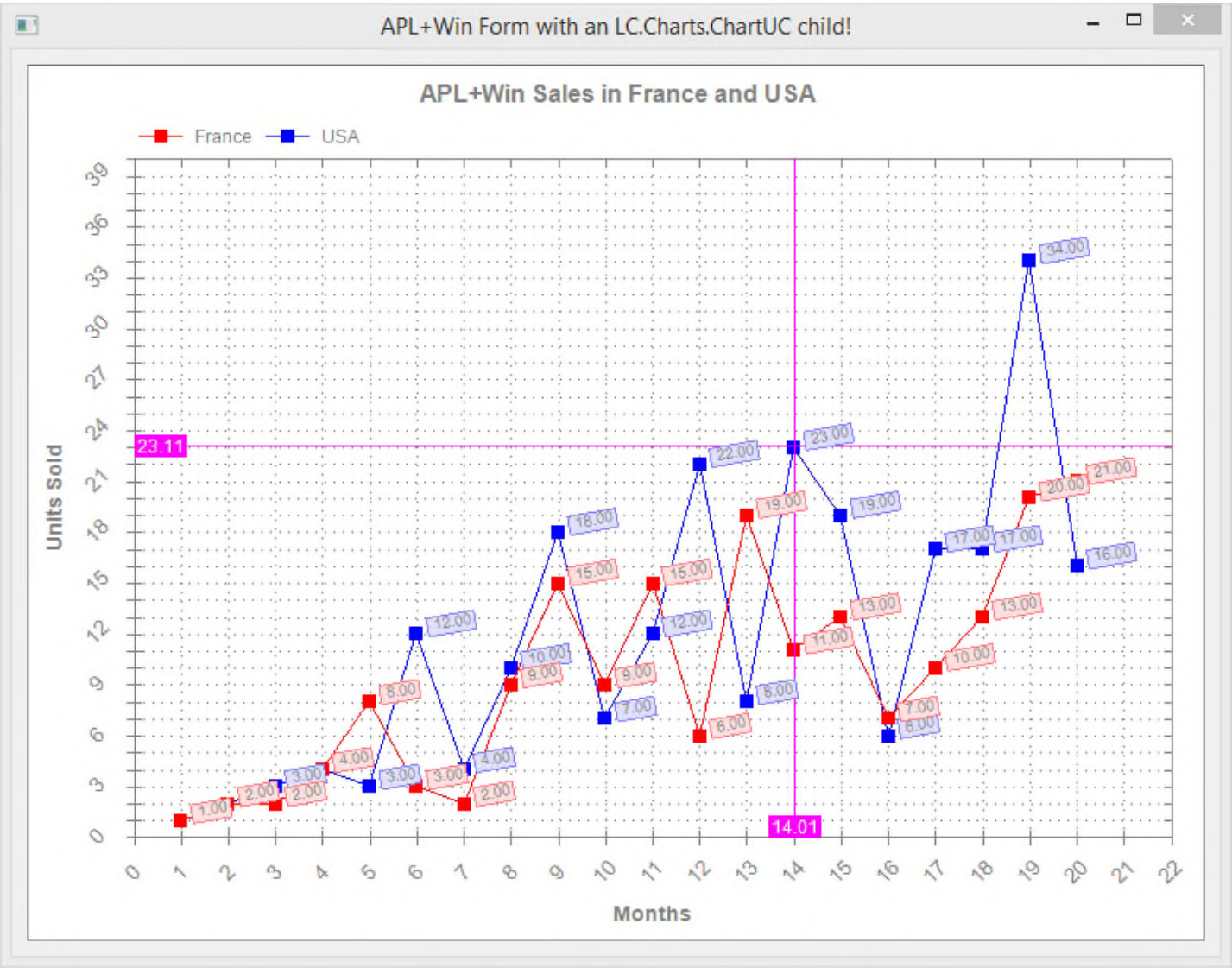
In the case of the above example, the TestFn5 function is:

```
    ∇ TestFn5 a;c;d;data;z;⎕wself
[1]   ⍝∇ TestFn5 a -- Function generated by FormChart!
[2]
[3]   :select a
[4]   :case''
[5]       z←'ff'⎕wi'*Create' 'Form'('*style'16)('*scale'5)('*size'600 800)'*Hide'
[6]       z←'ff'⎕wi'*caption' 'APL+Win Form with an LC.Charts.ChartUC child!'
[7]       z←'ff'⎕wi'*.cc.Create' 'LC.Charts.ChartUC'('*xWhere'10 10 580 780)
[8]       z←⎕wcall'SetParent'('ff'⎕wi'*.cc.xHandle')('ff'⎕wi'*hwnd')
[9]       ⍝ Replace next line with your own data
[10]      z←'ff'⎕wi'*.cc.xData'(20 3⍴1 1 1 2 2 2 3 2 3 4 4 4 5 8 3 6 3 12 7 2 4
             8 9 10 9 15 18 10 9 7 11 15 12 12 6 22 13 19 8 14 11 23 15 13 19 16
             7 6 17 10 17 18 13 17 19 20 34 20 21 16)
[11]      z←'ff'⎕wi'*.cc.xBorder'1
[12]      z←'ff'⎕wi'*.cc.xCrossHairCursor'1
[13]      z←'ff'⎕wi'*.cc.xAllowDragPoint'1
[14]      ⍝ Replace next line with your own data⍵
[15]      z←'ff'⎕wi'*.cc.xLegend'('France' 'USA')
[16]      z←'ff'⎕wi'*.cc.xPointLabels'1
[17]      z←'ff'⎕wi'*.cc.xPointLabelsAngle'10
[18]      z←'ff'⎕wi'*.cc.xSymbol' 'square'
[19]      z←'ff'⎕wi'*.cc.xTitle' 'APL+Win Sales in France and USA '
[20]      z←'ff'⎕wi'*.cc.xXAngle'45
[21]      z←'ff'⎕wi'*.cc.xXAxisTitle' 'Months'
[22]      z←'ff'⎕wi'*.cc.xXMajorStep'1
[23]      z←'ff'⎕wi'*.cc.xXMax'22
[24]      z←'ff'⎕wi'*.cc.xXMin'0
[25]      z←'ff'⎕wi'*.cc.xXMinorStep'1
[26]      z←'ff'⎕wi'*.cc.xYAngle'45
[27]      z←'ff'⎕wi'*.cc.xYAxisTitle' 'Units Sold'
[28]      z←'ff'⎕wi'*.cc.xYMajorStep'1
[29]      z←'ff'⎕wi'*.cc.xYMax'40
[30]      z←'ff'⎕wi'*.cc.xYMin'0
[31]      z←'ff'⎕wi'*.cc.xYMinorStep'1
[32]      z←'ff'⎕wi'*.cc.XRedraw'
[33]      z←'ff'⎕wi'*.cc.XShow'
[34]      ⍝ Events
[35]      z←'ff'⎕wi'*onResize' 'TestFn5"onResize"'
[36]      ⍝ Show (or Wait)
[37]      z←'ff'⎕wi'*Show'
[38]  :case'onResize'
[39]      (c d)←⎕wi'*size'
[40]      z←⎕wi'*.cc.xWhere'10 10,c d-20
[41]  :endselect
    ∇
```

All this is basic, but allows you to quickly and easily create an APL function that produces the chart you want without having to remember all the property names you want to use.

Just run:

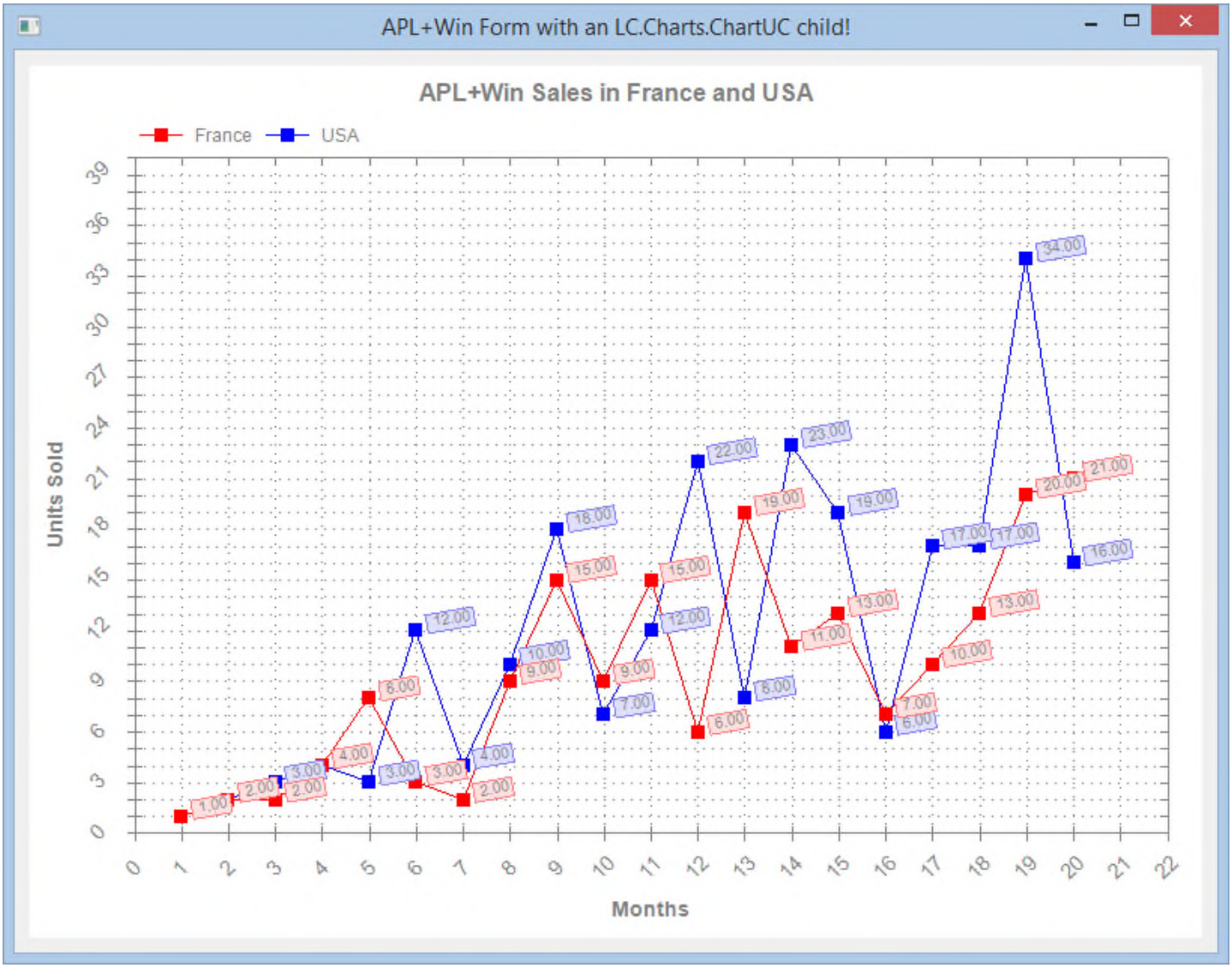
```
TestFn5''
```



and it creates an APL Form with the embedded LC.Charts chart you wanted to get. From there, you can easily edit the generated function to make some final adjustments.

For example, I finally did not like the chart border and the cross hair cursor, so I just edited the TestFn5 function and changed these properties to 0 and here is how the chart now look:

```
TestFn5''
```



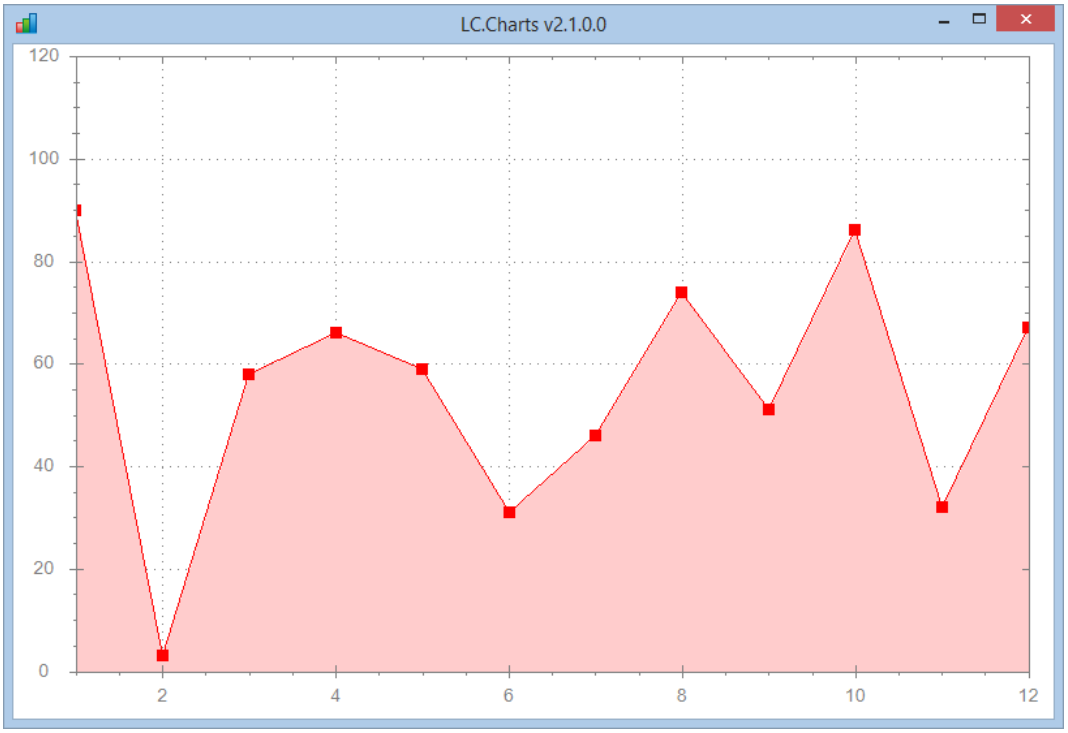
Curve Points Drag and Drop (new in v2.1.0.0)

In version 2.1, you can now drag and drop curve points which result in your curves being changed.

This option is disabled by default, but you can enable it by setting the new **xAllowDragPoint** property to **1**.

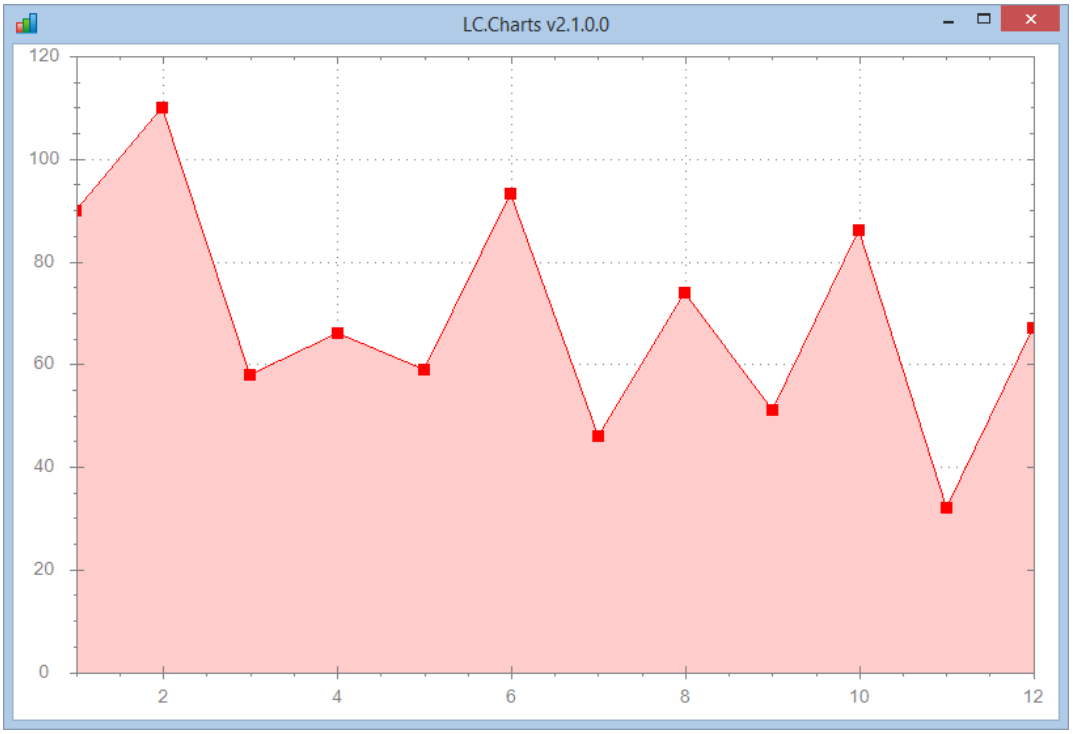
Here is an example:

```
]chart ?12⍴100 /cf=80 /sy=square /ymax=120 /ymin=0
```




```
'lccharts'⎕wi'*xAllowDragPoint'1  ⍝ allow drag and drop points with Alt + Mouse Left
```

Now drag points at abscissa 2 and 6 to a much higher position (remember to use the Alt key): you get:



Updating xData when points are dragged (new in v2.3.0.0)

Starting with v2.3.0.0, the points coordinates are automatically updated in the **xData** property when you drag and drop points.

This poses a little difficulty when the X values are (often consecutive) integers since you may drag a point to a (slightly) different X value than its original X integral value. You do not want in that case the X value to be changed to 3.0023752 for example.

In the case of integral X values, it has been decided to not update the X values and assume you just wanted to move the point vertically to some new ordinate.

Y values are always updated.

Example:

```
      ]chart (⍳5),[1.5]?5⍴5 /adp /sy=square /tr
      ⎕wi'*xData'
 1 4
 2 5
 3 3
 4 3
 5 5
```

After dragging and dropping point **2**, **3** and **5**, retrieving the **xData** property gives:

```
      ⎕wi'*xData'
 1 4
 2 4.3623931
 3 3.9245698
 4 3
 5 4.8172194
```

Even though at least one of the points (point 3) was dragged to an X position slightly different from 3 its X value remained 3.

This feature does not apply if you have provided floating point values for the X values.

Point Labels Drag and Drop (new in v2.1.0.0)

Starting with version 2.1, you can now drag and drop point labels. This can be useful to solve some overlapping problems whenyou have many points in your chart.

Just remember to use the **Alt** key and your Mouse Left Button to do the drag and drop operation.

This feature is available by default: you don't have to set any property to use it.

Localization (new in v2.1.0.0)

It is now possible to localize LC.Charts.

All you need to do is to call the **XLocalizeTo** method, specifying your culture[12].

The supported cultures are:

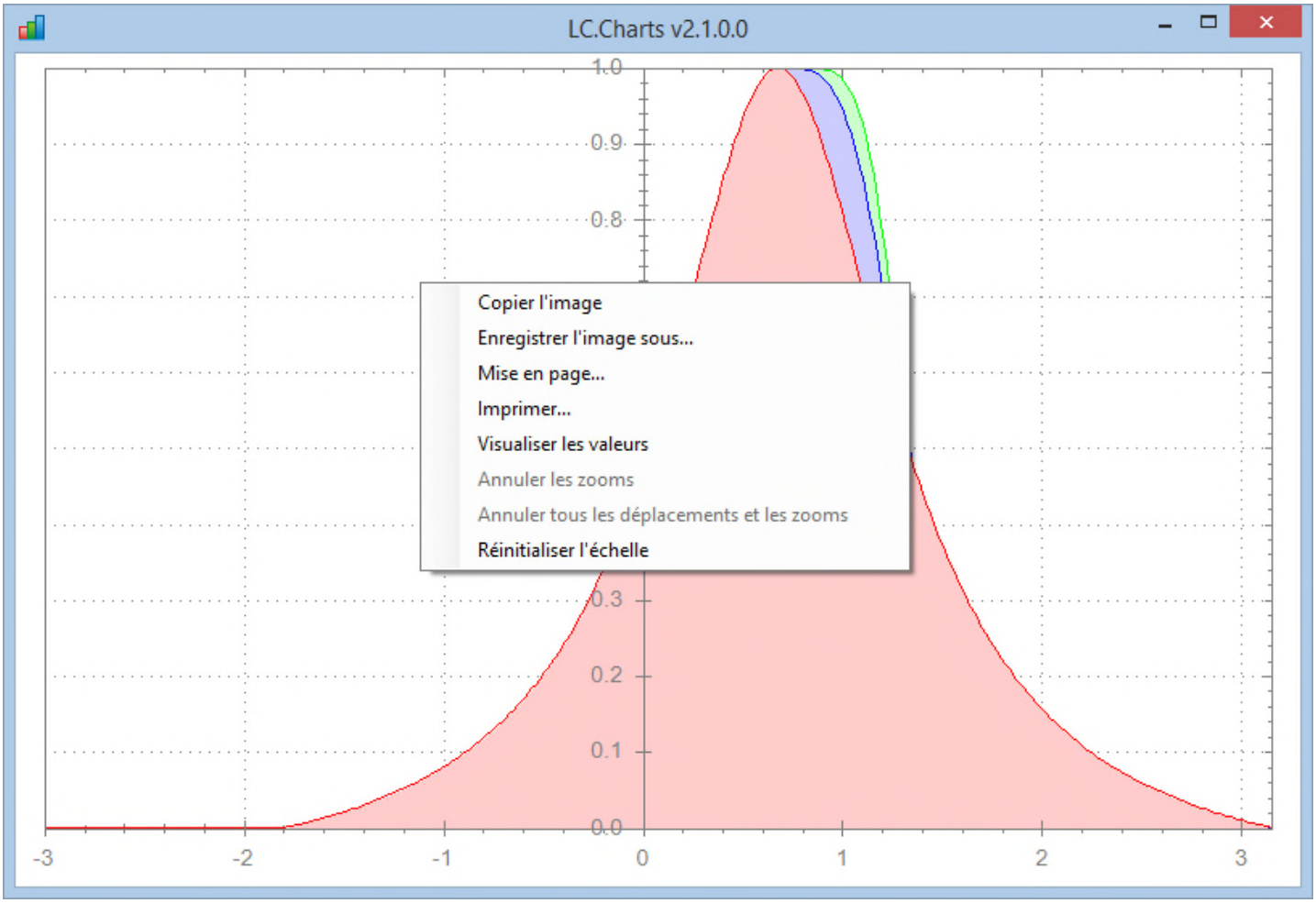
- **en-US** English (USA)
- **de-DE** German (Germany)
- **es-ES** Spanish (Spain)
- **fr-FR** French (France)
- **hu-HU** Hungarian (Hungary)
- **it-IT** Italian (Italy)
- **ja-JP** Japanese (Japan)
- **pt-PT** Portuguese (Portugal)
- **ru-RU** Russian (Russia)
- **sk-SK** Slovak (Slovakia)
- **sv-SE** Swedish (Sweden)
- **tr-TR** Turkish (Turkey)
- **zh-CN** Chinese (China)
- **zh-TW** Chinese (Taïwan)

[12] A list of all cultures can be found at : http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx

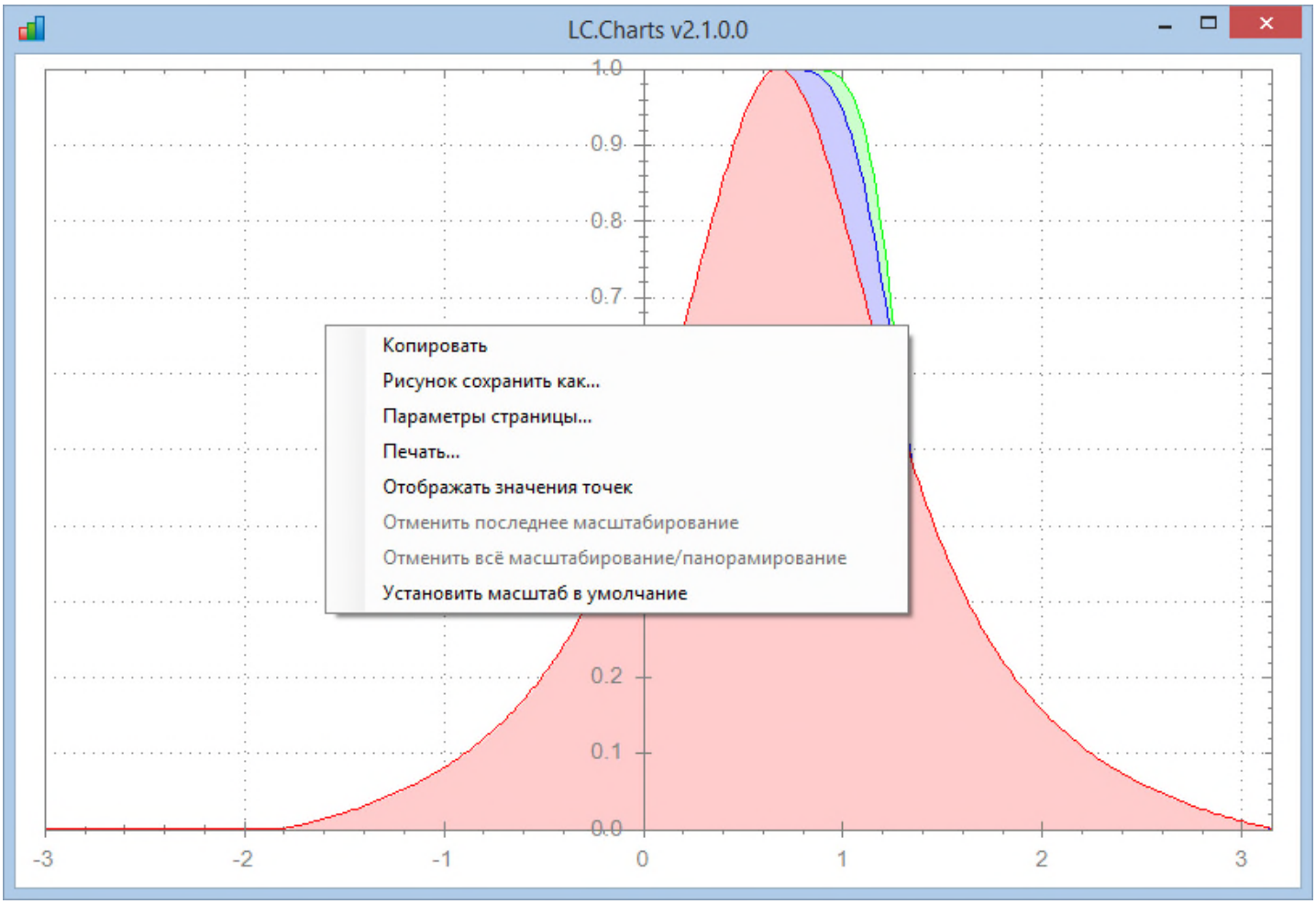
Example:

```
]chart SampleData1 /cf=80

'lccharts'wi'*XLocalizeTo' 'fr-FR'
```



```
'lccharts'wi'*XLocalizeTo' 'ru-RU'
```



The Euro to Dollar conversion rates since beginning of Euro (new in v2.2.0.0)

Here is an example of a real data very simple curve using LC.Charts.Chart.

It uses the new **XEuroDollarRates** method (in the LC.Charts.Chart object only) to retrieve up to 100 rates between 2 dates provided as YYYYMMDD values.

Example:

```
      'ff'⎕wi'*Create' 'LC.Charts.Chart'
ff

      'ff'⎕wi'*XEuroDollarRates'20131015 20140122
 20141015 1.2666
 20141016 1.2749
 20141017 1.2823
 20141018 1.275974
 20141019 1.275974
 20141020 1.2773
 20141021 1.2762
 20141022 1.2693
 20141023 1.2669
 20141024 1.2659
 20141025 1.2670385
 20141026 1.2670385
 20141027 1.2679
 …
```

The XEuroDollarRates C# method uses a Web Service[13] to retrieve the Euro to Dollar conversion rates

So the following expression allows to draw the evolution of the rates since 15 october 2014:

```
      ]chart (2+DateBase data[;1]),(data←'lccharts'⎕wi'*XEuroDollarRates'20141015
20150123)[;,2] /xty=date
```

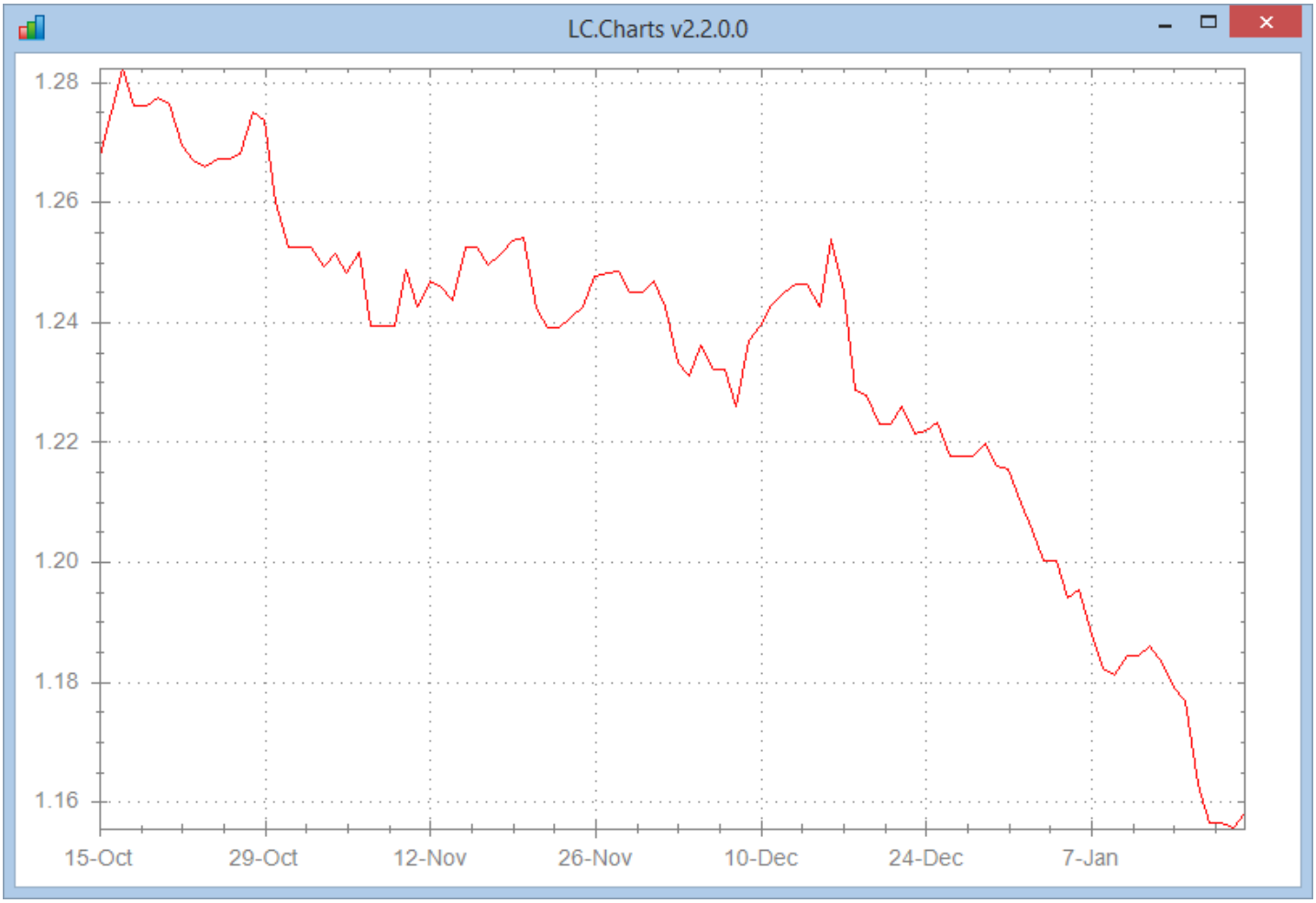
This expression is interesting as:

1. It shows that you can pass complex APL expressions to the]chart command
2. It even shows that you can make use of the lccharts[14] instance of LC.Charts.Chart within the command itself

[13] At : http://currencies.apps.grandtrunk.net

Note the use of the **date** X-Axis type (/xty=date), but in that case you must use the **DateBase** APL utility with an offset of 2 days to convert the **YYYYMMDD** dates returned by **XEuroDollarRates** to number of days since Jan. 1, 1900.

```
      ]chart (2+DateBase data[;1]),(data←'lccharts'⎕wi'*XEuroDollarRates'20141015
20150123)[;,2] /xty=date
```



Web Services like the one at http://currencies.apps.grandtrunk.net generally do not allow you to retrieve more than **100** rates at a time, and since a Web Service is never fast, I cheated to get the complete curvve since the beginning of the euro.

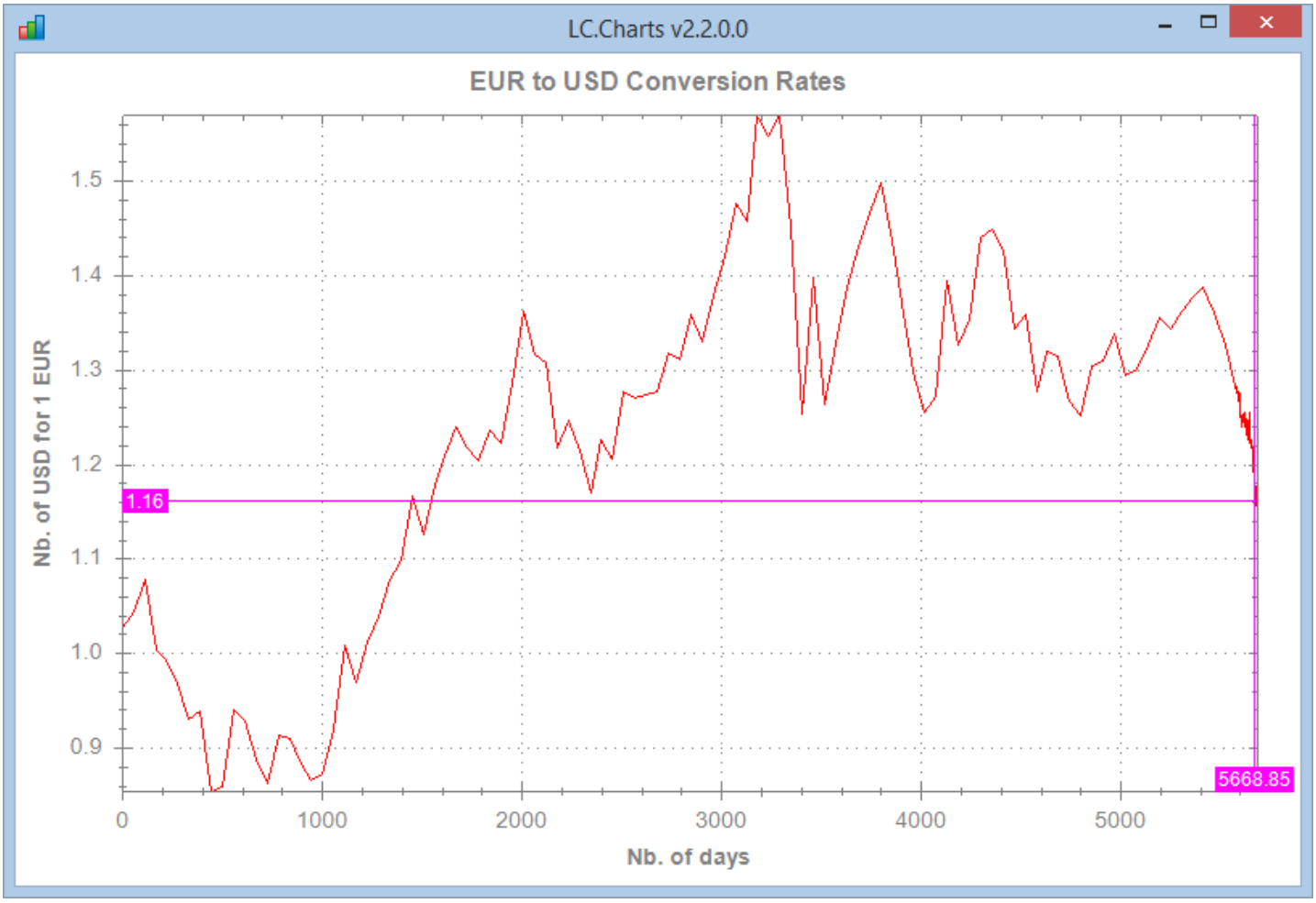
The **EuroDollarRates** APL function delivered in the **LCCHARTS** user command file:

1. Retrieves 100 rates from beginning of Euro till 100 days ago (i.e. approximately 1 rate every 56 days)
2. Retrieves 100 rates (one per day) for the last 100 days till today

The curve is not perfectly accurate but it still gives a pretty good idea of the rates evolution:

[14] lccharts is the name of the LC.Charts.Chart instance created by the]chart command

```
EuroDollarRates
```



Today (Saturday January 23, 2015) 1 € is indeed close to $1.16.

Alternative 2D Charts (new in v2.5.0.0)

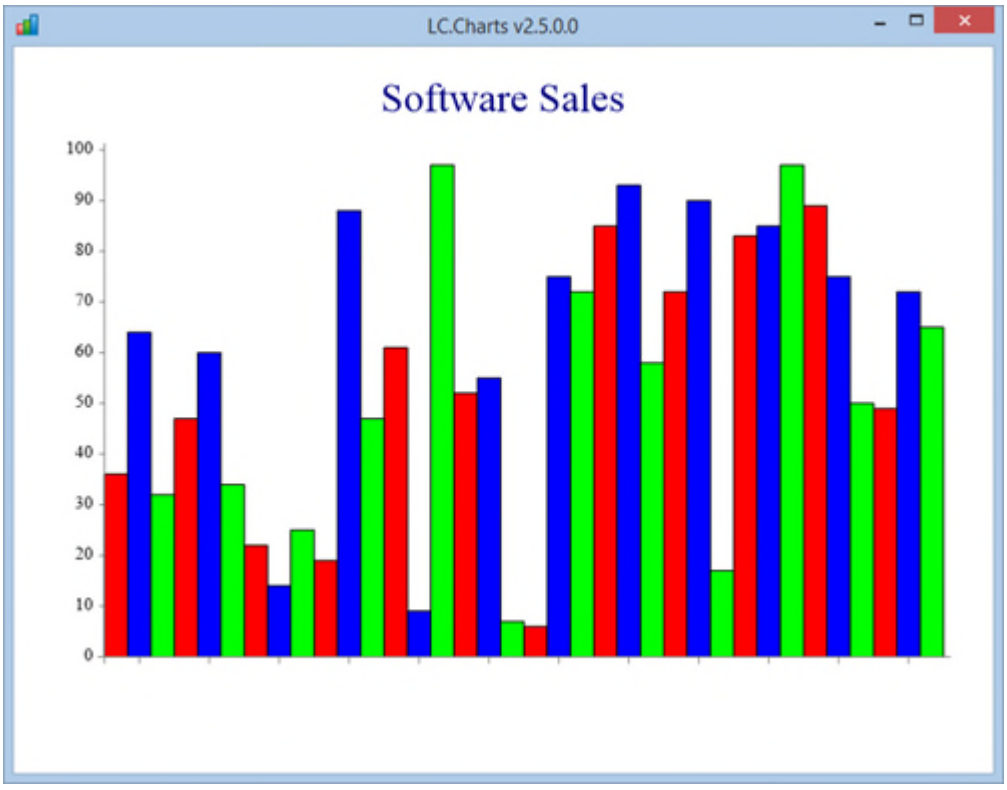
Starting with v2.5.0.0, **LC.Charts.Chart3D** includes the following alternative 2D charts types:

- **bar** (all kinds of Bar Charts)
- **curve** (Time Series Charts with or without trend lines)
- **polar** (Polar/Radar Charts)

Even though these are 2D charts, they are used by instantiating the **LC.Charts.Chart3D**[15] object or by using the **]chart3d** User Command.

Examples (new in 2.5.0.0)

```
]chart3d ⎕split⍉?10 3⍴100 /ti=Software Sales /ty=bar /re /to
```



This is the exact equivalent of the following instructions:

```
⎕wself←'lccharts3d'⎕wi'*Create' 'LC.Charts.Chart3D'
⎕wi'*xType' 'bar'
⎕wi'*xData'(⎕split⍉?12 3⍴100)
```

[15] The LC.Charts.Chart object can also create bar and curve charts: in general the LC.Charts.Chart object is significantly faster and should be used when you have a lot of points in your charts (like the Brent3 APL function that draws 200001 points!) and in general the LC.Charts.Chart3D object allows more flexibility and options in your charts.

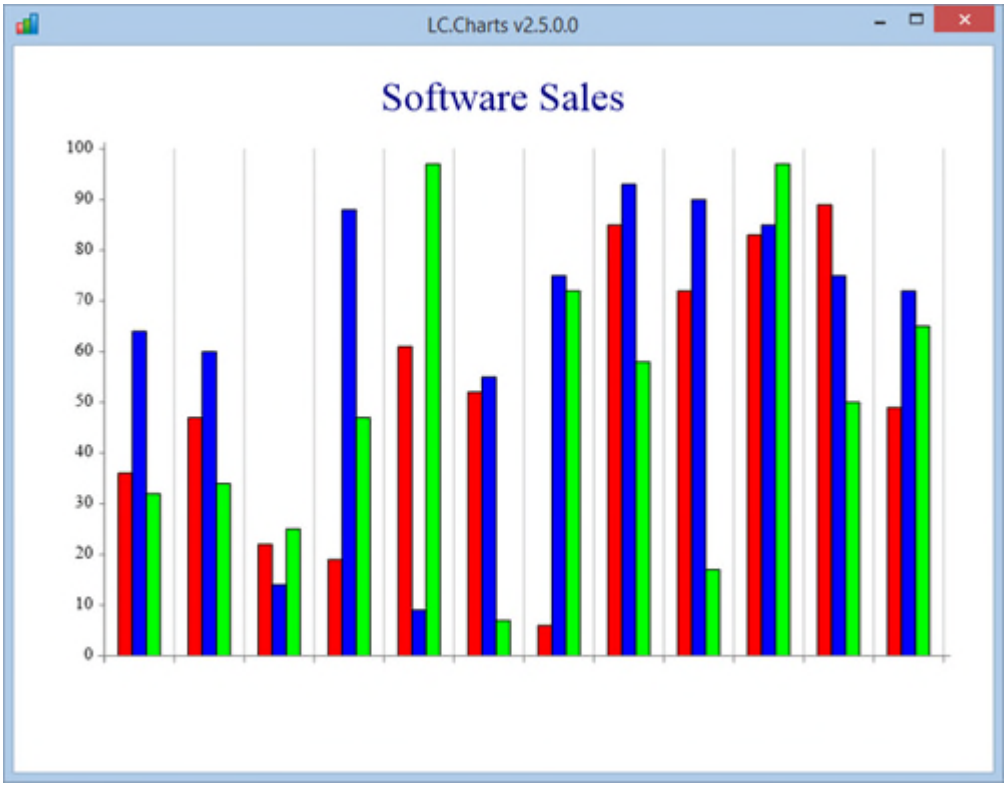
```
⎕wi'*xTitle' 'Software Sales'
⎕wi'*xTopMost'1
⎕wi'*xReflect'1
⎕wi'*XShow'
```

Once you have easily created the chart with:

```
]chart3d ⎕split⍉?10 3⍴100 /ti=Software Sales /ty=bar /re /to
```

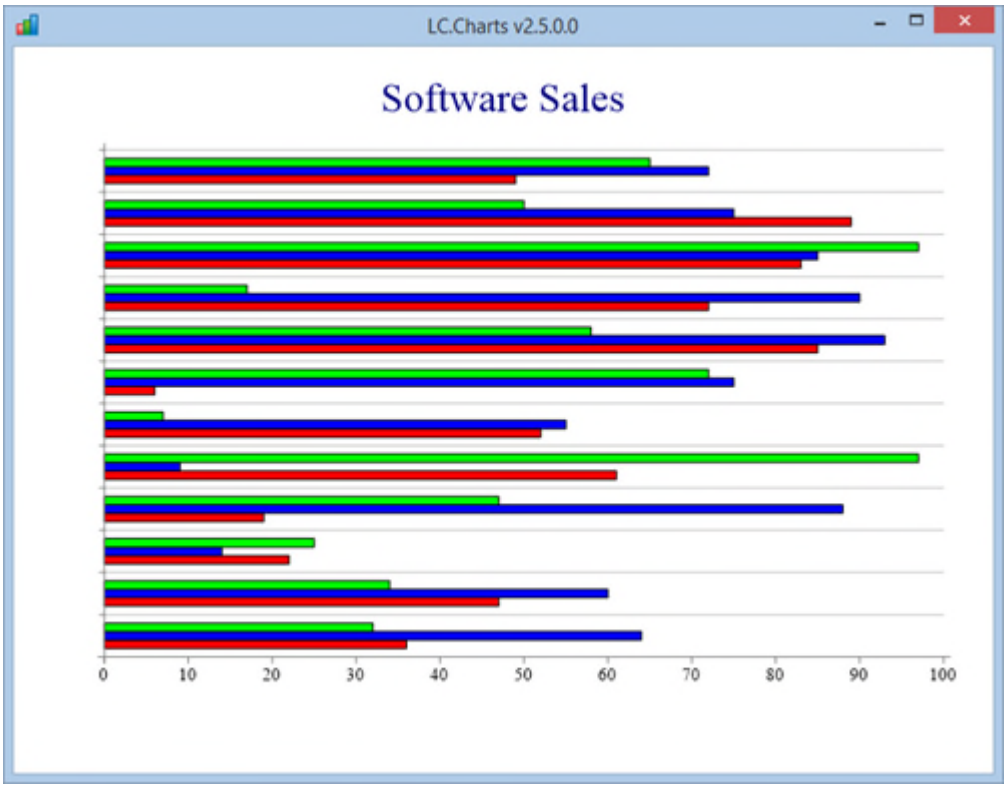
you can customize it, using ⎕wi:

```
⎕wi'*xGroupGap'2
⎕wi'*xBarChartStyle' 'TicksBetween'
⎕wi'*xXAxisStyle' 'GridLines'
```

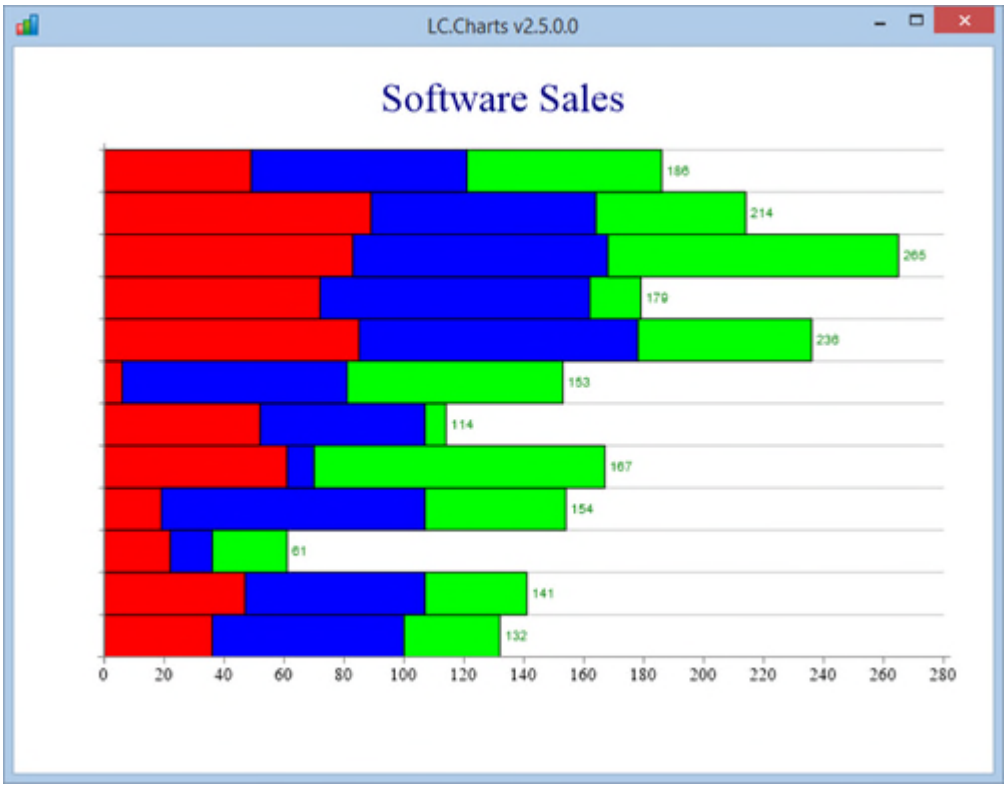



```
⎕wi'*xBarChartStyle' 'Horizontal,TicksBetween'
⎕wi'*xXAxisStyle' ''
```

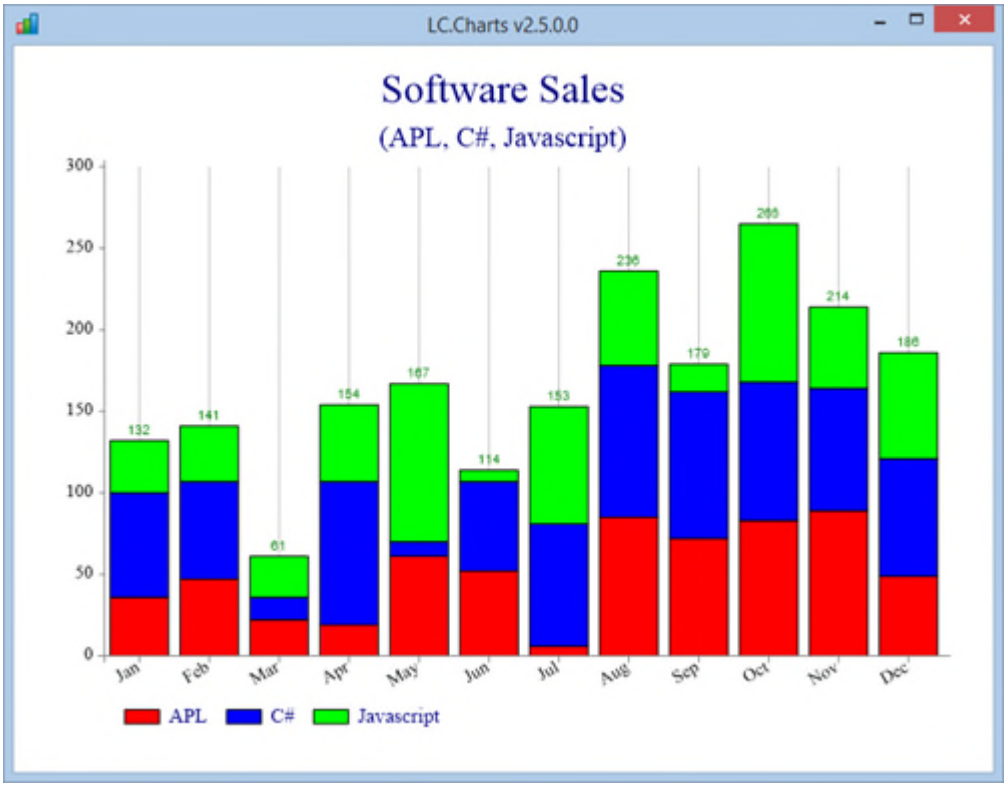
```
wi'*xYAxisStyle' 'GridLines'
```




```
wi'*xBarChartStyle' 'Horizontal,TicksBetween,StackedBars,ValueTags'
```



```
      ⍞wi'*xBarChartStyle' 'StackedBars,ValueTags'
      ⍞wi'*xYAxisStyle' ''
      ⍞wi'*xXAxisStyle' 'GridLines'
      ⍞wi'*xGap'.2
      ⍞wi'*xSubTitle' '(APL, C#, Javascript)'
      ⍞wi'*xTopMargin'50
      ⍞wi'*xLegend' 'APL' 'C#' 'Javascript'
      ⍞wi'*xXLabels' 'Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun' 'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec'
      ⍞wi'*xXAngle'30
```



Creating a StackedBars 100% chart is a little demanding.

You must adapt your data so that the sum of each row totals 100.

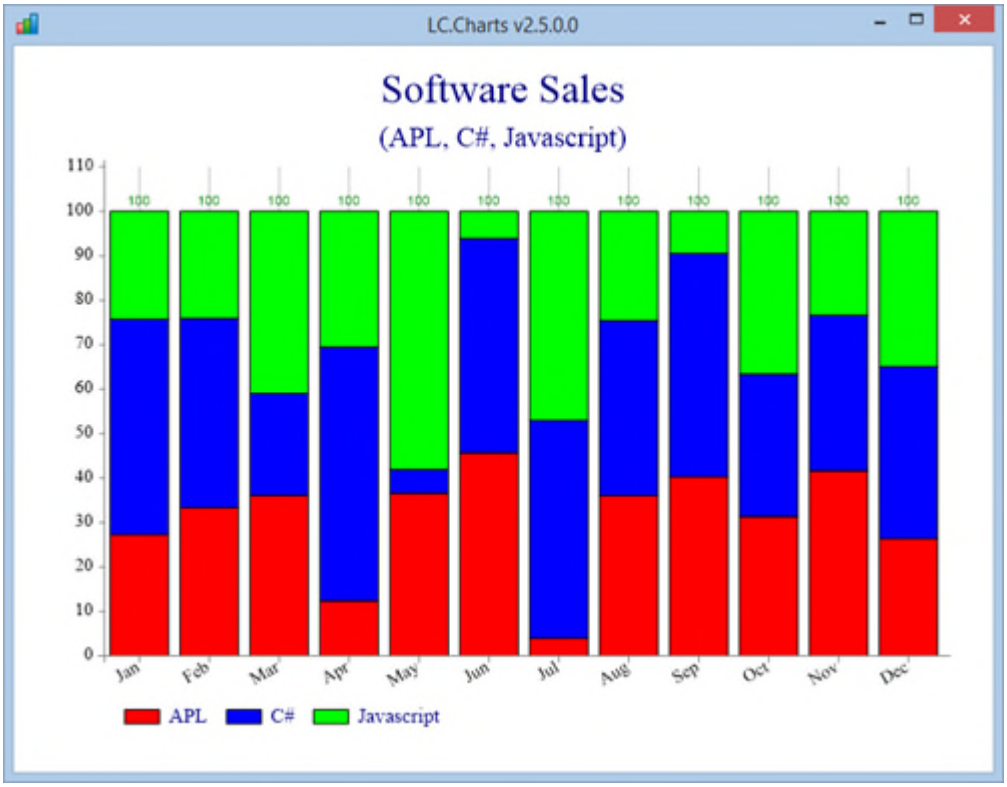
Assuming the original data is:

```
      data
 36 64 32
 47 60 34
 22 14 25
 19 88 47
 61  9 97
 52 55  7
  6 75 72
 85 93 58
 72 90 17
 83 85 97
```

```
 89 75 50
 49 72 65
```

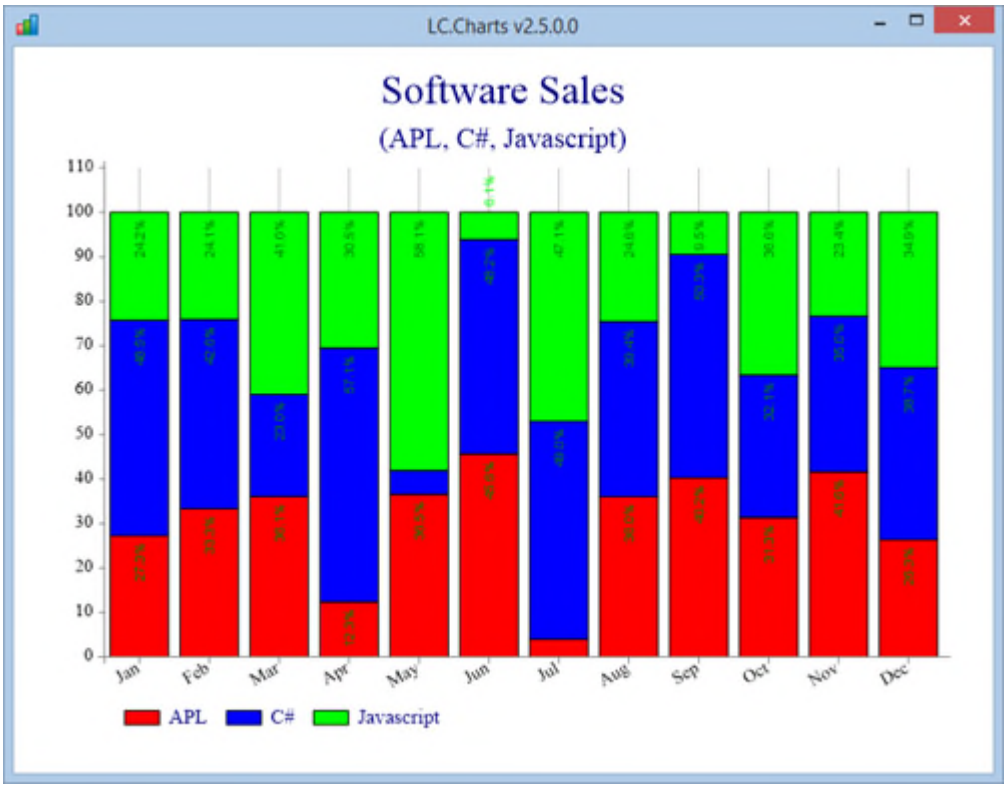
This is easily done using the **RndMat** (and **RndVec**) supplied utilities as follows:

```
      stackedBar100Data←100×17 RndMat data÷(⍴data)⍴(¯1↑⍴data)/+/data
      ⎕wi'*xData'(⎕split⍉stackedBar100Data)
```



Value tags can be displayed in each stacked bar as follows:

```
      ⎕wi'*xValueTagStyle' 'Vertical,Inside,SectorValues,RecolorOutside'
      ⎕wi'*xValueTagFormat' '#.0%'
```



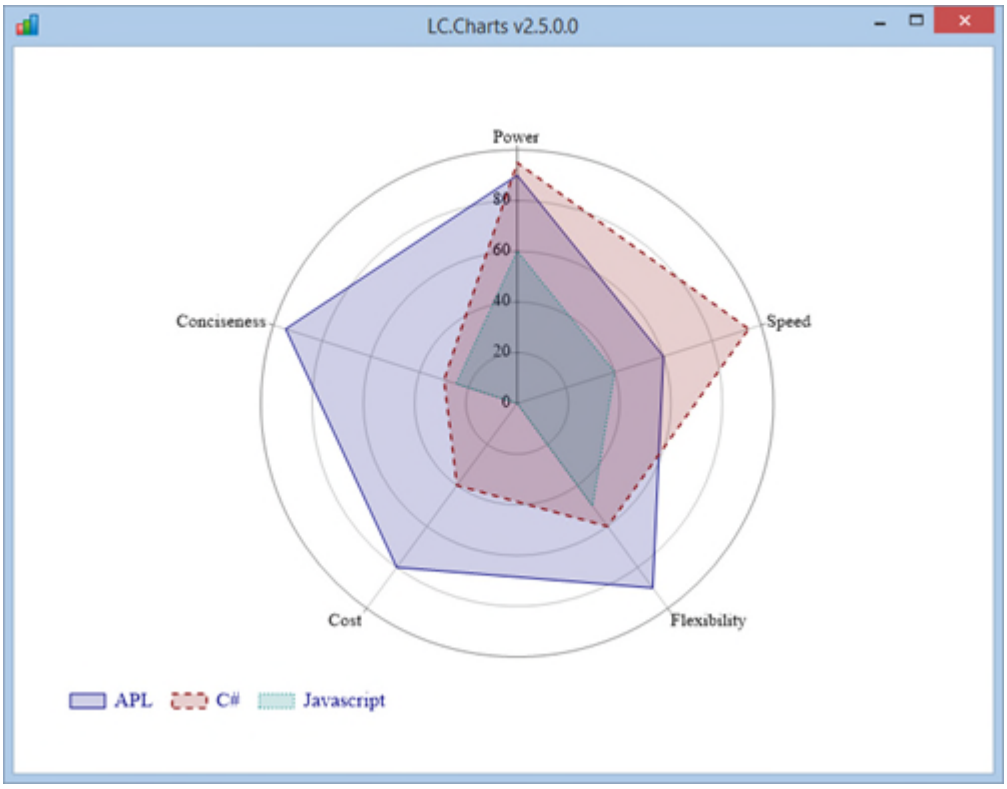
The principles are the same for creating **curve** charts.

You can create a **polar** chart as follows:

```
data←(90 60 90 80 95) (95 95 60 40 30) (60 40 50 0 25)

]chart3d data /ty=polar

⎕wi'*xLegend' 'APL' 'C#' 'Javascript'
⎕wi'*xXLabels' 'Power' 'Speed' 'Flexibility' 'Cost' 'Conciseness'
```



New sample APL functions (new in v2.5.0.0)

The **LCCharts.sf** User Command file contains several new sample functions to demonstrate the **bar**, **curve** and **polar** charts:

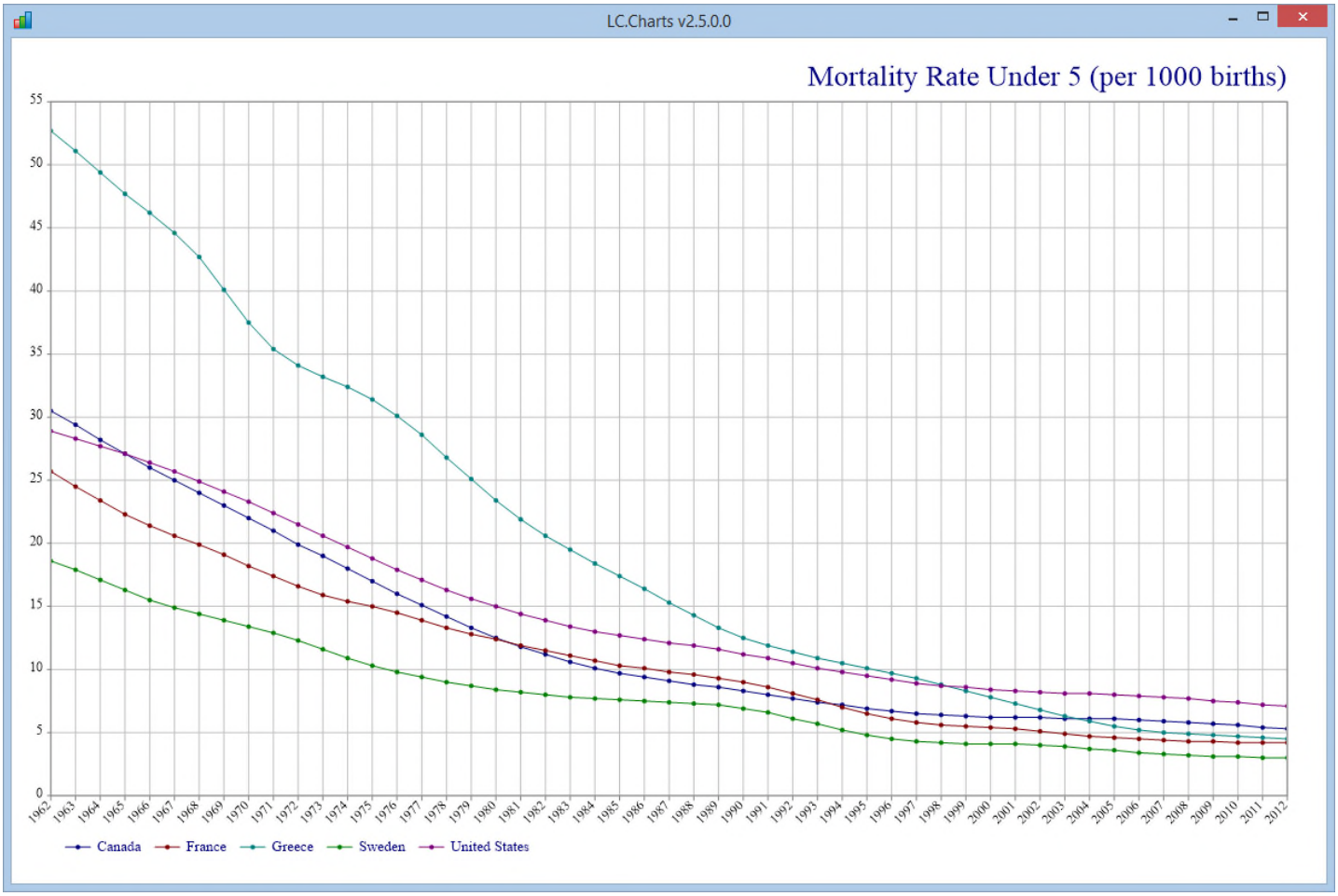
- **SampleBars**
- **SampleCurves**
- **SamplePolar**
- **SampleTrendCurves**

All these functions are niladic functions, so just type their names to execute them in the APL Session. Each time you execute one of these functions, the chart data, parameters, styles are randomly selected, so you can execute the same function a number of times to see the various type of charts you can create.

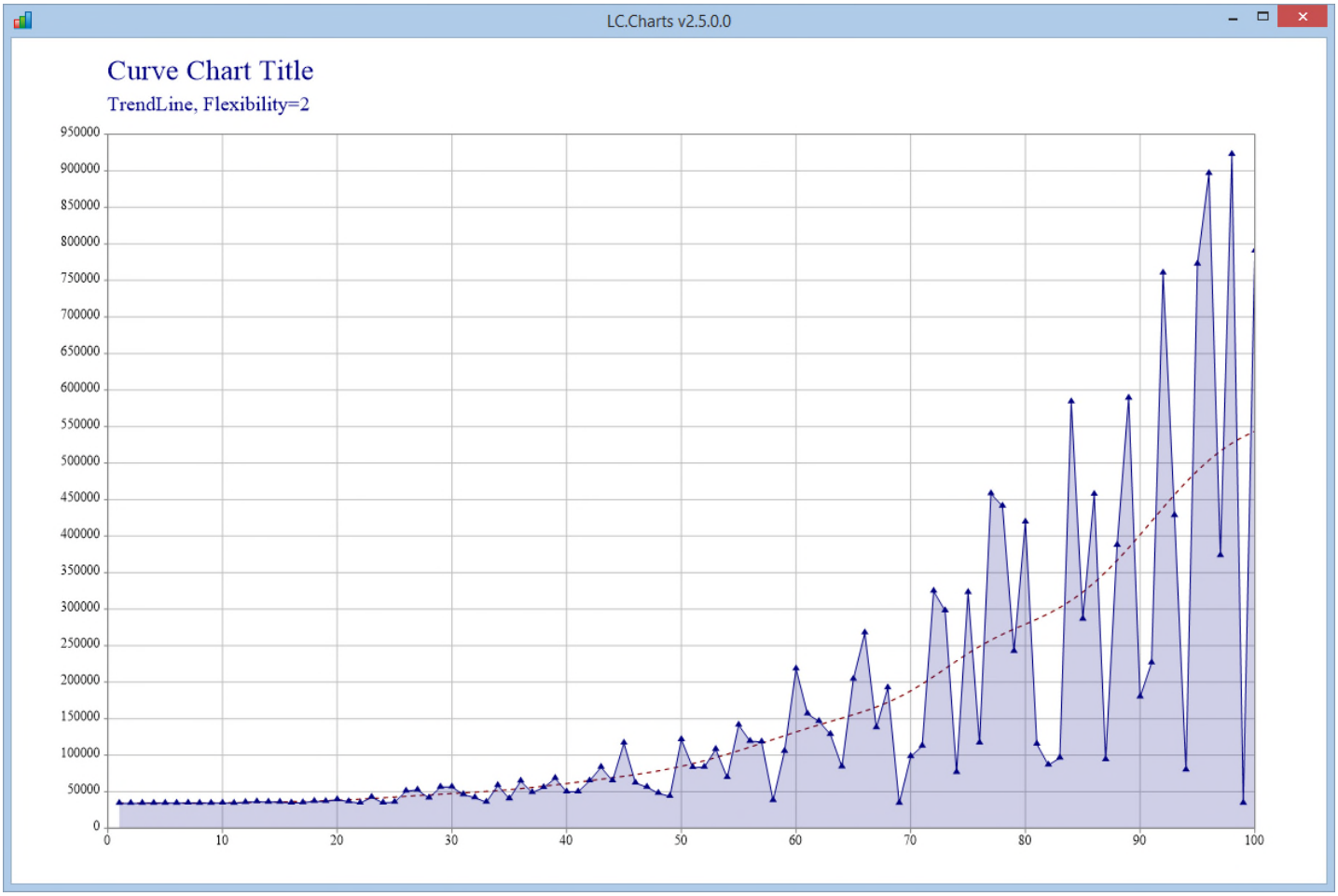
Remember that you can simply close the C# form displaying the chart by pressing **Escape**. This is very convenient as you can run one of these functions, then press Escape, then run it again, etc.

Example:

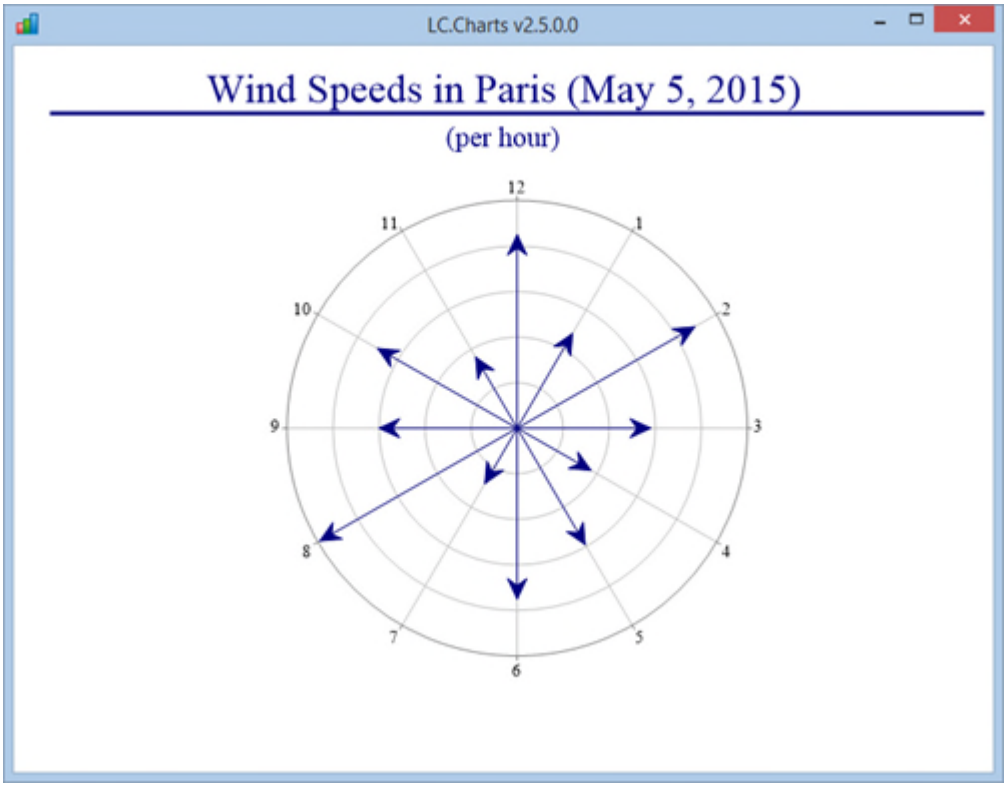
```
SampleCurves
```



SampleTrendCurves



```
SamplePolar
```



New properties and methods (new in v2.5.0.0)

A number of properties have been added in version 2.5.0.0. They are described below:

General Information Properties (new in v2.5.0.0)

The following general information read-only properties have been added:

- **xChartColors**
- **xChartSymbols**
- **xChartTypes**

```
      ⎕wi'xChartColors'
255 0 0   0 0 255   0 255 0   0 0 43   255 26 184   255 211 0   0 87 0   131 131 255
      158 79 70   0 255 193   0 131 149   0 0 123   149 211 79   246 158 219   211 17
      255   123 26 105   246 17 96   255 193 131   35 35 8   140 167 123   246 131 8
      131 114 0

      ⎕wi'xChartTypes'
```

```
 altitude bar curve fitsurface flattower planes polar response scatter
      simplesurface tiledsurface tower trendsurface

      ⎕wi'xChartSymbols'
 ball bar block bullet circle cross del diamond dot downtick hyphen invisible
lefttick lozenge node plus righttick ring smallbullet triangle uptick
```

Style properties (new in v2.5.0.0)

The following new style properties have been added:

- **xBarChartStyle**
- **xBarChartStyles**
- **xFillStyle**
- **xHeadingStyle**
- **xHeadingStyles**
- **xLineGraphStyle**
- **xLineGraphStyles**
- **xLineStyle**
- **xLineStyles**
- **xPolarChartStyle**
- **xPolarChartStyles**
- **xValueTagStyle**
- **xValueTagStyles**

Each of the **xxxxxStyle** property has an associated **xxxxxStyles** property.
You can retrieve the **xxxxxStyles** property value to know the possible styles you can apply:

```
      80 TELPRINT ⊃[2]⎕wi'xBarChartStyles'
CroppedAxes   ForceZero     Horizontal    NoAxes        TicksBetween
ExplodeAxes   FrameAxes     Indexed       OverlayGrid   ValueTags
FloatingBars  GridLines     Logarithmic   StackedBars
```

You apply some styles by separating them with a comma and by using the xxxxxStyle property: example:

```
      ⎕wi'*xBarChartStyle' 'Horizontal,NoAxes,FrameAxes,ExplodeAxes'
```

The xGap and xGroupGap properties (new in v2.5.0.0)

The **xGap** property is used to separate bars in a bar chart.
The **xGroupGap** property is used to separate bar groups in a side by side bar chart.

The xLegend and xLegendVisible properties (new in v2.5.0.0)

Two new properties are available:

- **Legend**
- **LegendVisible**

You can add a legend to a Chart3D chart by setting the **xLegend** property as a nested vector of vectors:

```
      ⎕wi'*xLegend' 'APL' 'C#' 'Javascript'
```

The legend can be made visible or invisible using the Boolean **xLegendVisible** property.

```
      ⎕wi'*xLegendVisible' 0
```

The new Axis related properties (new in v2.5.0.0)

The following properties have been added :

- **xXangle**
- **xXLabelFormat**
- **xYLabelFormat**
- **xZLabelFormat**
- **xXMin**
- **xXMax**
- **xYMin**
- **xYMax**

Use the **xXangle** property to force X-axis labels to be displayed with an angle: xXAngle must be expressed in degrees (0 to 360). Note that there is no xYAngle or xZAngle properties available.

Use the **xXLabelFormat**, **xYLabelFormat** or **xZLabelFormat** to set the various axis labels format.

Formats are to be expressed as C# format strings[16]. However the % symbol has no effect on the scale of the data. You can also use the ~ symbol anywhere in the format string to suppress the digit at that position.

Use the **xXMin**, **xXMax**, **xYMin** and/or **xYMax** properties to force the chart to start and/or end at a given X or Y value.

The new Value Tags related properties (new in v2.5.0.0)

The following new properties have been added:

- **xValueFont**
- **xValueTagFormat**

If you want to see value tags being displayed in your Chart3D **bar** or **curve** charts, you must include the **ValueTags** style in the **xBarChartStyle** or **xLineGraphStyle** property: example:

```
      ⎕wi'*xBarChartStyle' 'Horizontal,TicksBetween,StackedBars,ValueTags'
```

The **xValueFont** property must be expressed similarly as the APL ⎕wi **font** property, i.e.:

```
      ⎕wi'*xValueFont' 'Calibri' 10 1(255 0 0)
```

[16] See http://www.cheat-sheets.org/saved-copy/msnet-formatting-strings.pdf

The **xValueTagFormat** is to be expressed as a C# string with the same rules as the **xXLabelFormat** (see above).

The new Curve related properties (new in v2.5.0.0)

The following properties, used to customize curves, are new in v2.5.0.0:

- **xLineWidth**
- **xFlexibility**
- **xMissingValue**
- **xSymbol**

Use the **xLineWidth** property to set the width of the curves. If you have 3 curves you must set the xLineWidth property as a floating point scalar or as a 3-element floating point vector.

You can now choose the symbols drawn at points on your curves by setting the **xSymbol**[17] property to either a single symbol name in which case it applies to all curves on the graph or to a comma delimited string of symbols: example:

```
□wi'*xSymbol' 'diamond'
□wi'*xSymbol' 'diamond,ball,dot,invisible,lozenge'
```

Note that you can use **invisible** symbol to avoid displaying any symbol on a given curve.

Remember that you can use the **xSymbolSize** property to change the symbol sizes.

The **xFlexibility** property is only used when you include the **TrendLine** style in the **xLineGraphStyle** property. It represent the degree of smoothing[18] used for the trend line.

Setting the **xMissingValue** property is very handy when you want to skip these values rather than drawing a line to the missing value point (for example 0).

In the industry, missing values are often set to **-999.25**, but you can set xMissingValue to 0 to skip 0 values in your line charts if 0 is the missing value. The default value for xMissingValue is **32768**.

The new xScale property (new in v2.5.0.0)

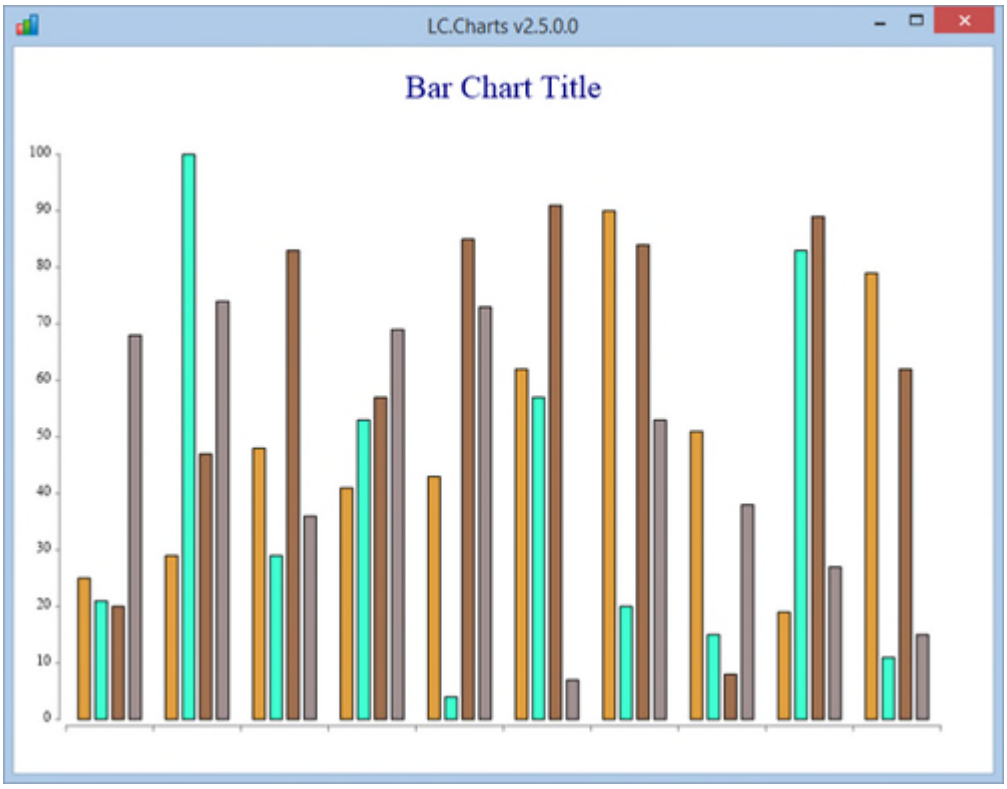
The new **xScale** property affects the size of labels in your Chart3D charts. The default **xScale** value is **1**.

This can, for example, be useful when you want to draw your chart in a maximized (or large) form: in that case you may find that the various texts are too large. By setting the **xScale** property to a value less than **1** you can reduce the size of the various textual elements.

```
SampleBars
```

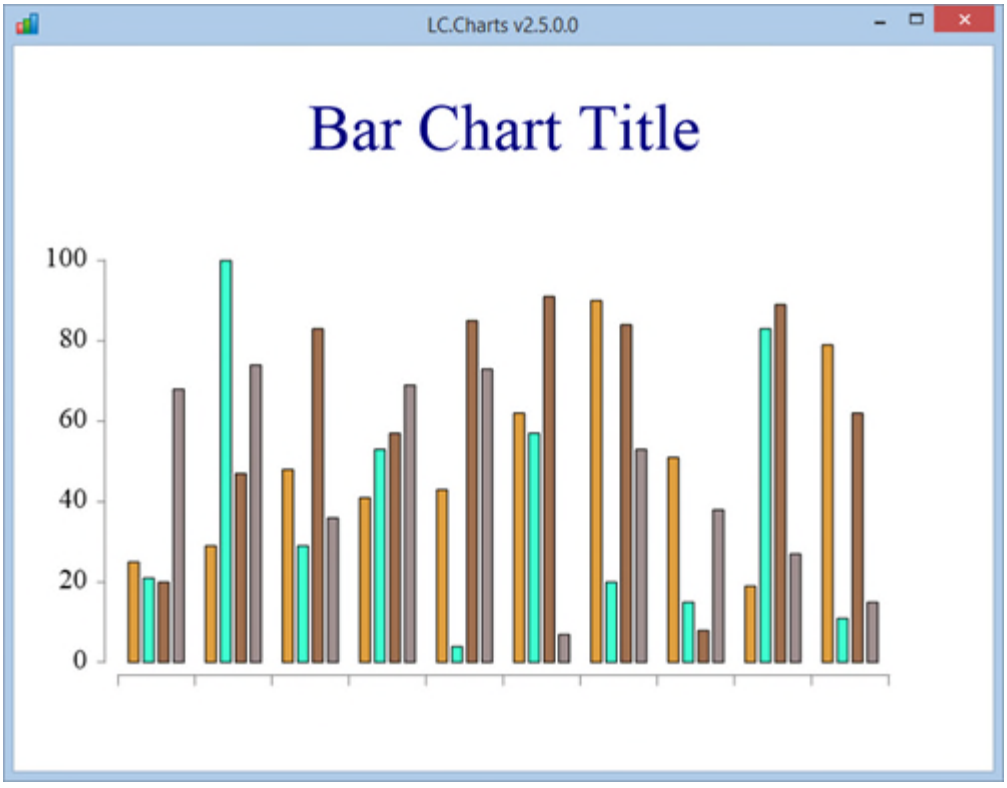
[17] See the **xChartSymbols** property to learn about the symbols you can use.

[18] Trend Lines are Gaussian-weighted moving averages. The 'Flexibility' sets the exponent of the gaussian, which is calculated by dividing the x-axis range arbitrarily into 100 parts and for each point by computing a y-value by adding up all the y-values from the data, weighted by the x-distance expressed as a classic 'bell-curve' function.



By setting the **xScale** property to a larger value you can force the labels to be larger (and reciprocally):

```
□wi'*xScale'2 ◇ □wi'*XRedraw'
```



3D Charts (new in v2.2.0.0)

Version 2.2.0.0 brings 3D Charts to LC.Charts.

This version is kind of experimental: it includes few properties for the 3D charts and is made for you to explore these charts. More properties will be added later to help you customize the 3D charts more.

3D Chart Types

LC.Charts v2.2.0.0 supports 10 different 3D chart types:

- **altitude** shading charts (see APL function: **Sample3DAltitude**[19])
- **fitsurface** charts (see APL function: **Sample3DFitSurface**)
- **flattower** charts (see APL function: **Sample3DFlatTower**)
- **planes** charts (see APL function: **Sample3DPlanes**)
- **response** plot charts (see APL function: **Sample3DResponse**)
- **scatter** plot charts (see APL function: **Sample3DScatter**)
- **simplesurface** charts (see APL function: **Sampel3DSimpleSurface**)
- **tiledsurface** charts (see APL function: **Sample3DTiledSurface**)
- **tower** charts (see APL function: **Sample3DTower**)
- **trendsurface** charts (see APL function: **Sample3DTrendSurface**)

These charts are presented below.

The new]chart3d user command (new in v2.2.0.0)

To produce these 3D charts, you can use the new **]chart3d** user command.

Run:

```
]chart3d?
```

for documentation about this command.

Version 2.2.0.0 includes a small subset of options for the 3D charts. More options will be added in later versions.

One difference with the **]chart** user command is that you must supply data to **]chart3d** as a nested vector of numeric vectors, with the first element generally being the X values, the second the Y values and the remaining elements Z series values. If your data is stored in an APL matrix with X in column 1, Y in column 2, Z in column 3, …, you can easily transform it to become a suitable argument to **]chart3d** with the following APL expression:

```
⍉↓data
```

[19] Note: all the **Sample3Dxxxxxx** APL functions use an LC.Charts.Chart instance called: **lccharts3d**
Also, all these functions use random data, so you can call them several times with different results each time.

Note that, just like with]chart, you can progressively build your chart by trial and error, adding more and more options[20] and checking the result at each stage:

```
      ]chart3d ⎕split⍉20 3⍴?⍳100

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower/vp=⍪30 30 30

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower/vp=⍪30 30
30/zmaj=5

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower/vp=⍪30 30
30/zmaj=5/zmax=40

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower/vp=⍪30 30
30/zmaj=5/zmax=35

      xlab←(⊂'Week '),¨⍕¨⍳20

      ylab←'APL' 'C#' 'VB.Net'

      ]chart3d ⎕split⍉20 3⍴?⍳100 /to/re/ri/ti=My Chart/ty=tower/vp=⍪30 30
30/zmaj=5/zmax=35/xl=⍪xlab/yl=⍪ylab
```

The new LC.Charts.Chart3D C# ActiveX object (new in v2.2.0.0)

The **LC.Charts.Chart3D** C# ActiveX object is a C# form containing a 3D chart object.

You can create an instance of it using:

```
      'lccharts3d'⎕wi'*Create' 'LC.Charts.Chart3D'
```

You then provide data to this object with the xData property:

```
      'lccharts3d'⎕wi'*xData' (⎕split⍉?20 3⍴⍳100)
```

And you finally display the chart using the **XShow** method:

```
      'lccharts3d'⎕wi'*XShow'
```

The default chart type used, if you don't specify an **xType** property value, is **fitsurface**.

[20] Please note that you don't have to use any space to separate your options: this makes it even easier to use]chart or]chart3d.

You can of course set a number of properties before you display your chart:

```
      TELPRINT⊃[2]'lccharts3d'⎕wi'*properties'
apldata       events        opened        xData         xXLabels
children      instance      pending       xHandle       xXMajorStep
class         interface     progid        xReflect      xYLabels
clsid         links         properties    xRisers       xYMajorStep
data          methods       self          xTitle        xZMajorStep
def           modified      state         xTopMost      xZMax
description   modifystop    suppress      xType         xZMin
errorcode     name          unicodebstr   xViewPoint
errormessage  obj           version       xWhere
```

Some of these properties are similar to the LC.Charts.Chart object (**xData, xHandle, xReflect, xTitle, xTopMost, xType, xWhere, xXMajorStep, xYMajorStep**) and some are specific to 3D charts (**xRisers, xViewPoint, xXLabels, xYLabels, xZMin, xZMax**).

Note that the properties which have the same names as the LC.Charts.Chart ones, should work the same.

For example you would use **xTopMost**, **xReflect** and **XRedraw** exactly as you do it with LC.Charts.Chart (see these properties and method documentation in the LC.Charts.Chart section).

The new LC.Charts.Chart3DUC C# ActiveX Control (new in v2.2.0.0)

Just as there is an **LC.Charts.ChartUC** control allowing you to embed **2D** charts in your APL+Win forms, there is a new **LC.Charts.Chart3DUC** control to let you embed **3D** charts in your APL+Win forms.

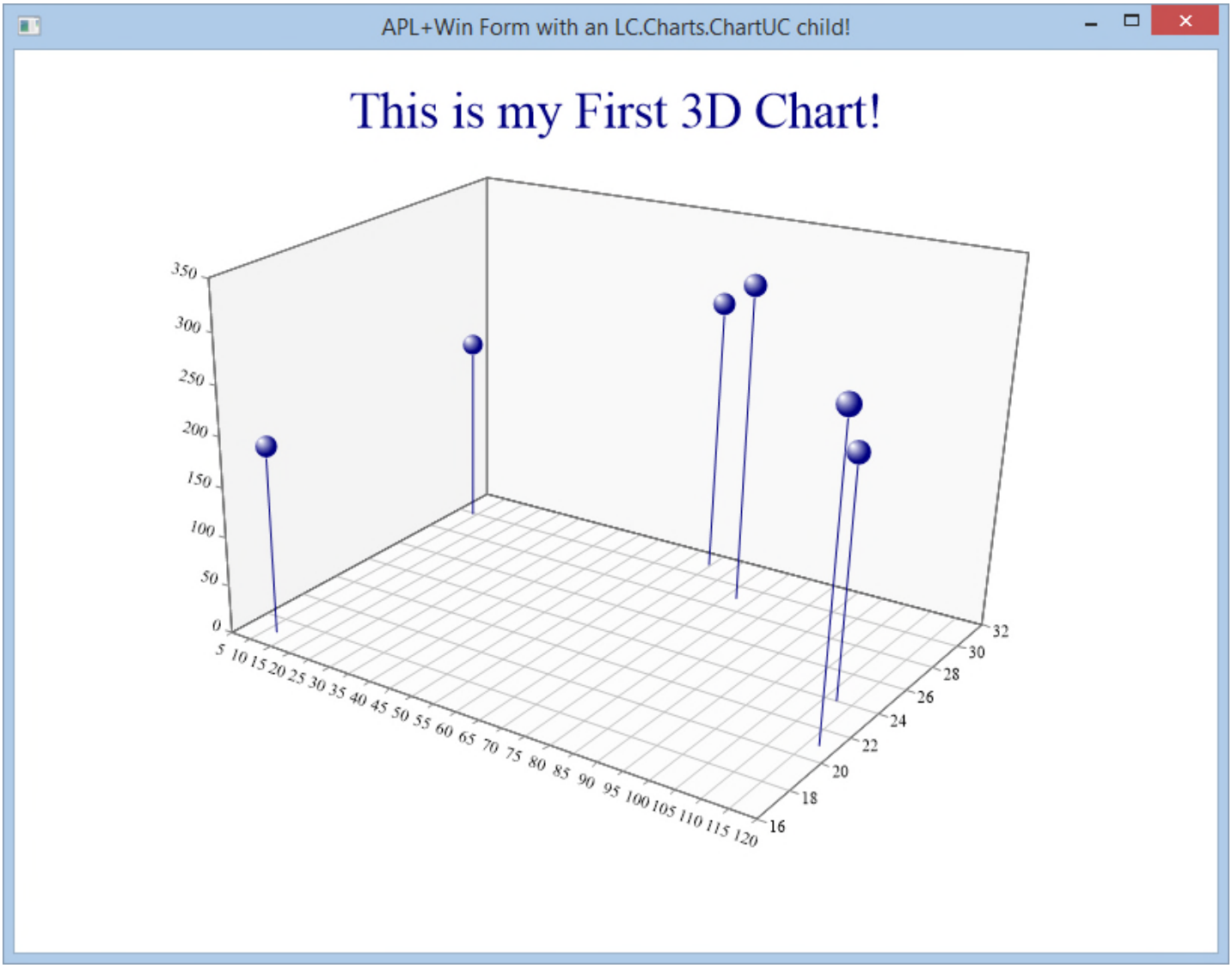
The **FormChartBasic3D** APL function demonstrates how you can embed a 3D chart in your APL+Win form:

```
    ∇ FormChartBasic3D a;c;d;x;y;z;⎕wself
[1]   ⍝∇ FormChartBasic3D a -- Creates an APL+Win Form with an LC.Charts.Chart3DUC
control
[2]   ⍝∇ (c) 2015 Lescasse Consulting
[3]   ⍝∇ ELE15jan15
[4]
[5]   :select a
[6]   :case''
[7]       z←'lccharts3d'⎕wi'*Create' 'Form'('*style'16)('*scale'5)'*Hide'
[8]       z←'lccharts3d'⎕wi'*caption' 'APL+Win Form with an LC.Charts.ChartUC child!'
[9]       z←'lccharts3d'⎕wi'*.cc.Create' 'LC.Charts.Chart3DUC'('*xWhere'10 10 580
780)('*xReflect'1)
[10]      z←⎕wcall'SetParent'('lccharts3d'⎕wi'*.cc.xHandle')('lccharts3d'⎕wi'*hwnd')
[11]
[12]      x←12,65,77,117,9,112
[13]      y←17,31,29,21,30,24
[14]      z←190,270,310,300,190,230
[15]      z←'lccharts3d'⎕wi'*.cc.xData'x y z
[16]      z←'lccharts3d'⎕wi'*.cc.xRisers'1
[17]      z←'lccharts3d'⎕wi'*.cc.xXMajorStep'5
[18]      z←'lccharts3d'⎕wi'*.cc.xTitle' 'This is my First 3D Chart!'
[19]
[20]      ⍝ Events
[21]      z←'lccharts3d'⎕wi'*onResize' 'FormChartBasic3D"onResize"'
[22]      z←'lccharts3d'⎕wi'*size'600 800
[23]      ⍝ Show the Form
[24]      z←'lccharts3d'⎕wi'*Show'
[25]  :case'onResize'
[26]      (c d)←⎕wi'*size'
[27]      z←⎕wi'*.cc.xWhere'¯3 ¯3,c d+6
[28]  :endselect
    ∇
```

The key is really to use the **SetParent** Win32 API function on line **10** to force the C# **Chart3DUC** User Control object to become a child of the APL+Win Form.

You can call this function as follows:

```
FormChartBasic3D''
```



The above example has the **Chart3DUC** object use the whole client area of the APL form, but you can of course display your 3D chart along other APL controls in your form (as is done in function **FormChart** with the **LC.Chart.ChartUC** C# User Control).

When a 3D chart is up and visible on the screen, you can animate it and spinit, doing[21]:

```
]spinit[22]
```

[21] This only works if your 3D chart object instance is named: **lccharts3d** (for an LC.Charts.Chart3D object) or **lccharts3d.cc** (for an LC.Charts.Chart3DUC object embedded in an APL+Win form)

[22] See the specific section later on in this document about Spin3D and]spinit for more details

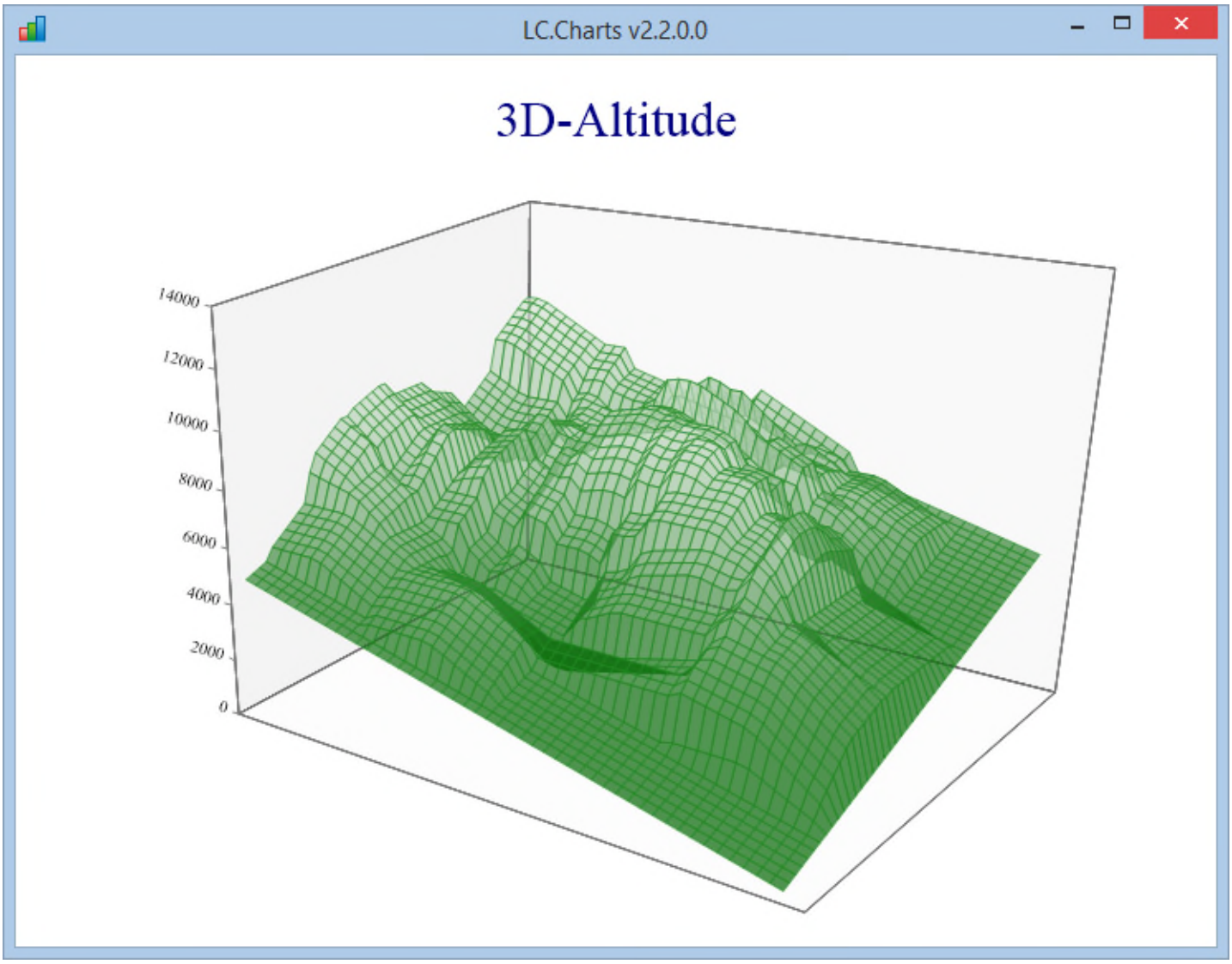
Altitude Shading Charts (/type=altitude)

This chart allows to display a surface with a shading proportional to the altitude.

The **Sample3DAltitude** APL functions demonstrates this chart. It uses a small APL utility called **Terrain** which can randomly produce a terrain with plains, hills, mountains represented by an APL matrix.

The result of **Terrain** is a suitable argument for the **]chart3d** user command.

```
Sample3DAltitude
```

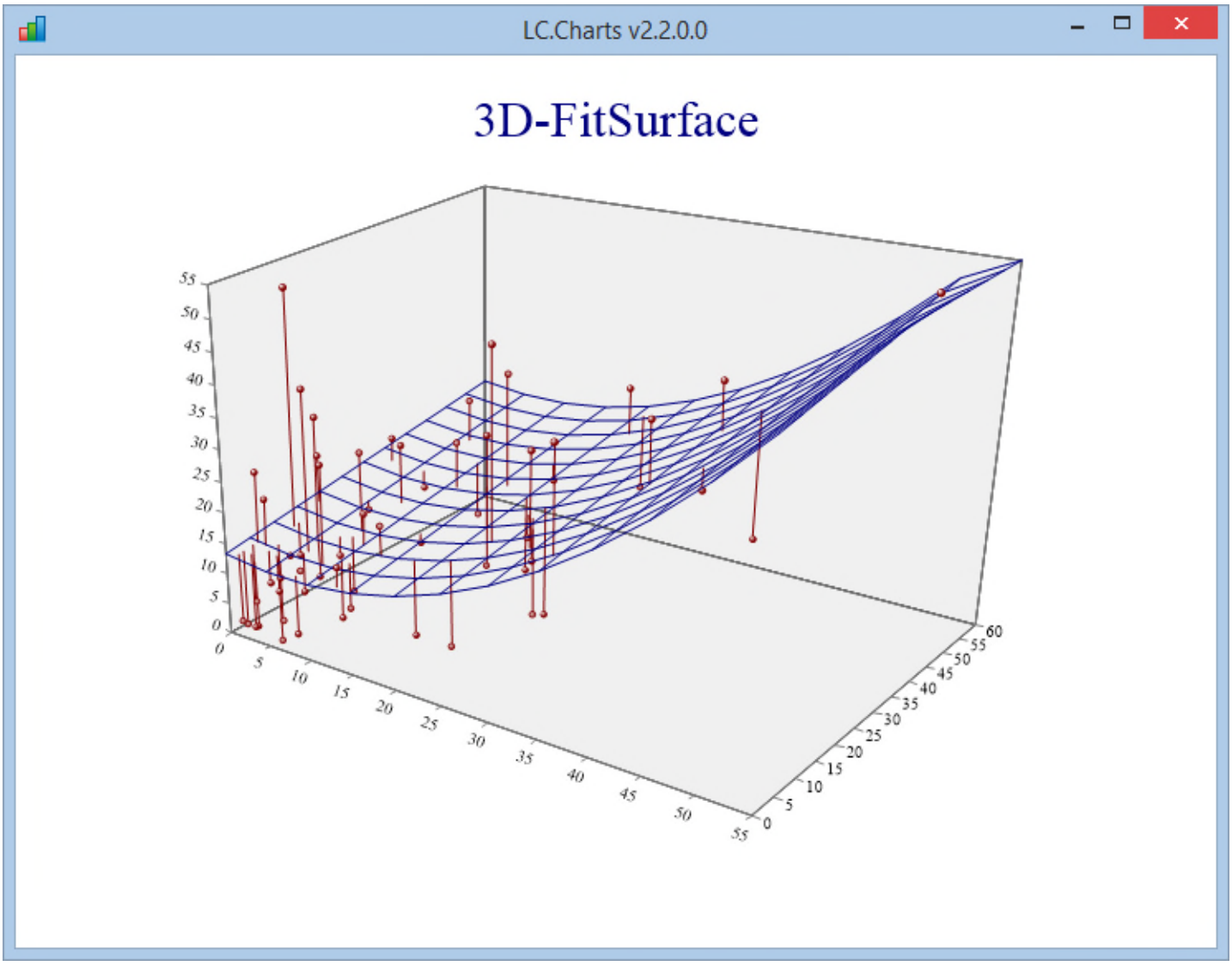


Fit Surface Charts (/type=fitsurface)

This plot fits a regression surface to a set of spatial points.

You can use the **Sample3DFitSurface** APL function to create examples of such a chart.

```
Sample3DFitSurface
```

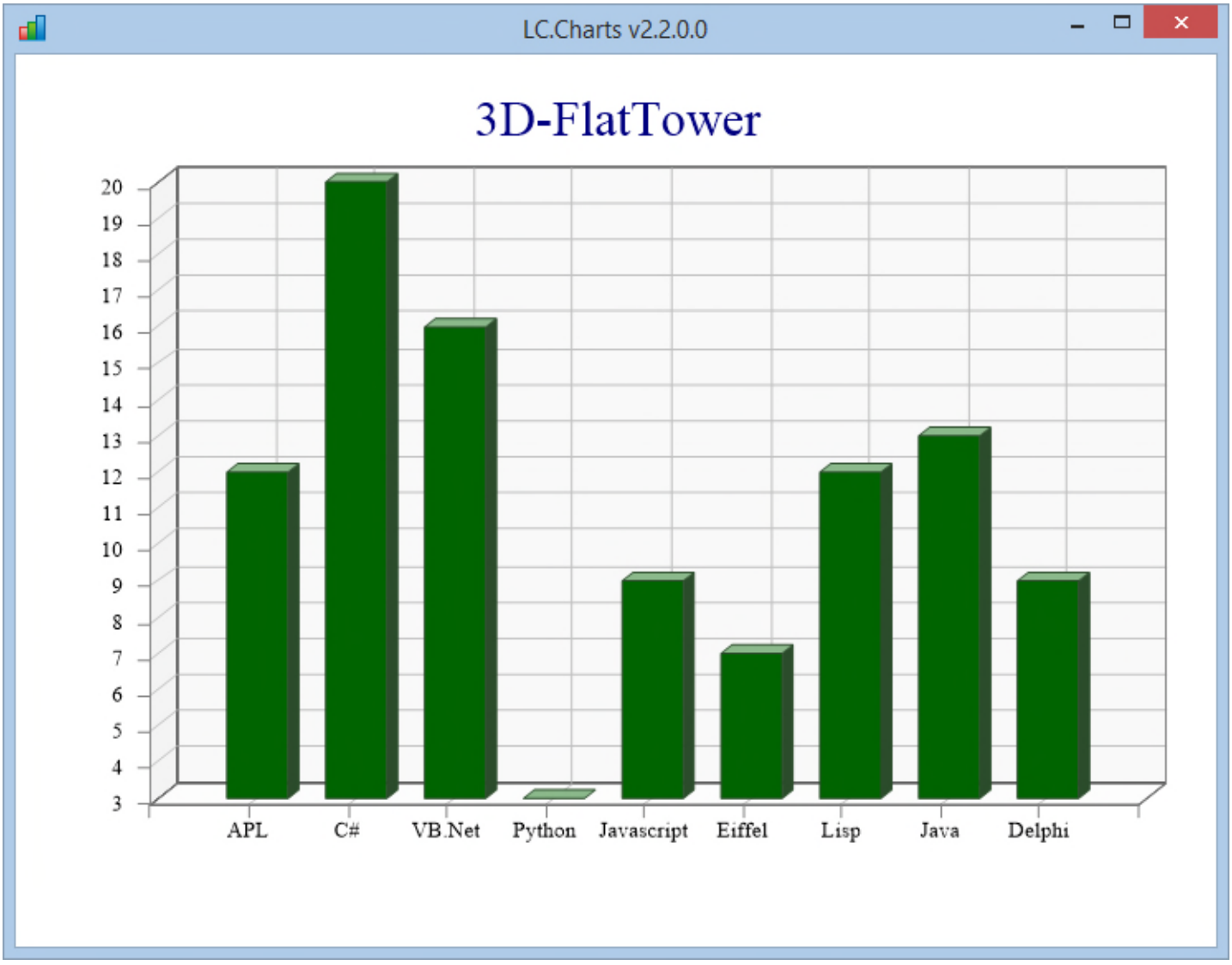


In LC.Charts v2.2.0.0, the model is quadratic on the X and linear on the Y values.

Flat Tower Charts (/type=flattower)

These towers are not really flat as they are 3D towers, but they are "flat" compared to the "**tower**" type of chart (see further one) and also because they are all displayed in the same plane.

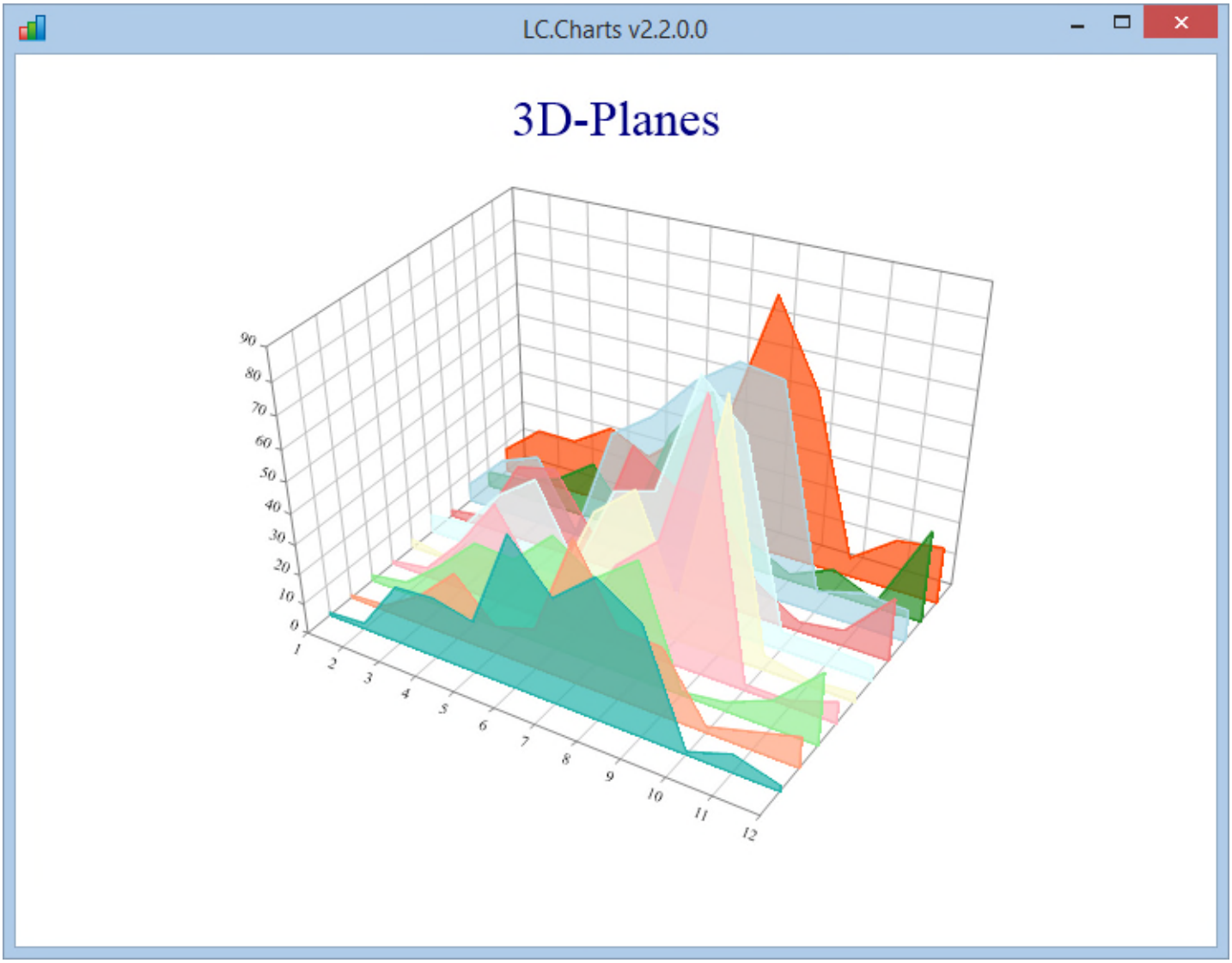
This chart is basically a bar chart with a little bit of a 3D effect.



Plane Charts (/type=planes)

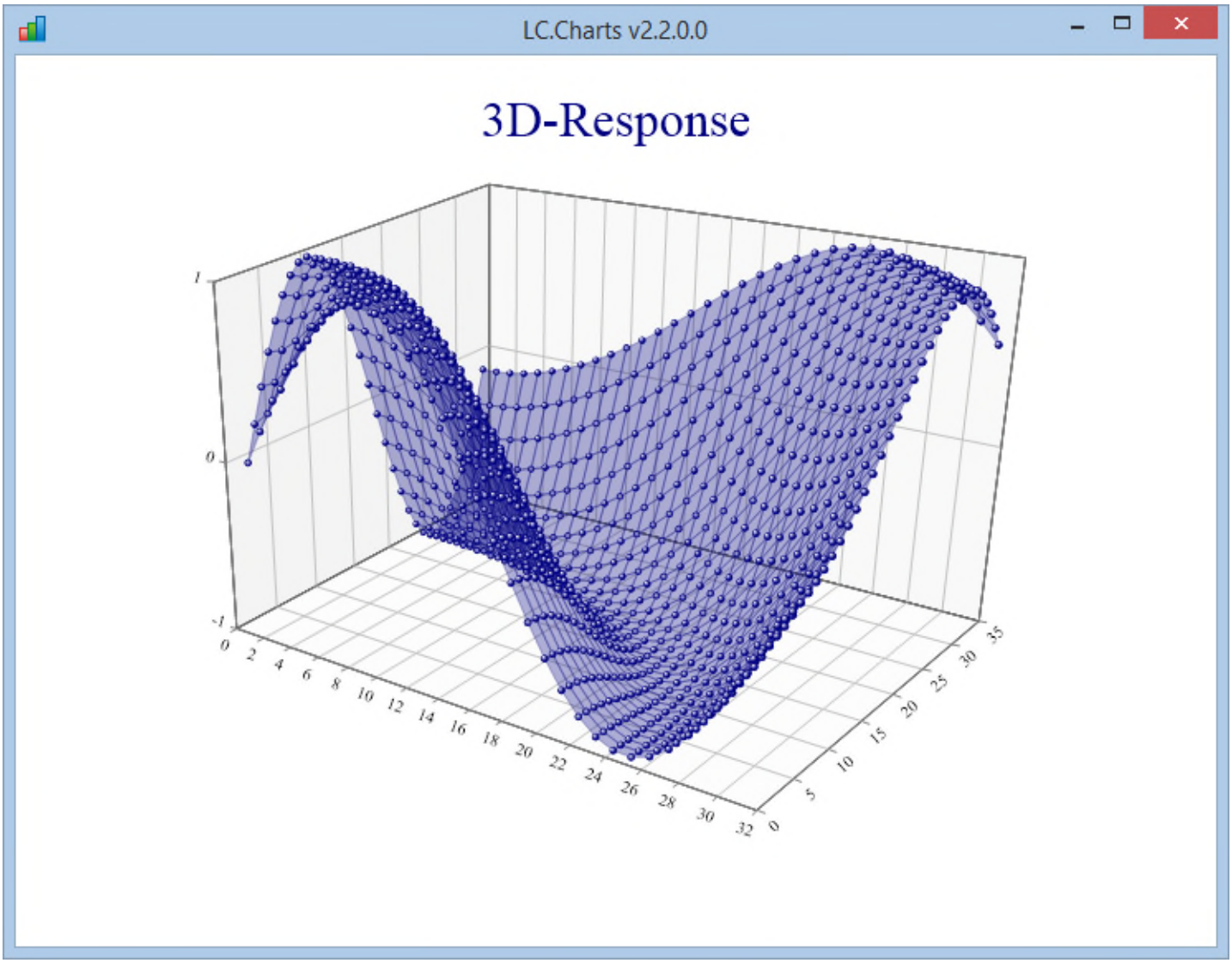
This chart is made for comparing any number of series, generally over time.

It is more readable if the number of series is small, but the transparency allows to (somewhat) see series through other series.



Response Plots Charts (/type=response)

This kind of chart is best suited to very regular surfaces, most of the time “mathematical” surfaces.



Displaying the true X and Y values in Response plots (new in v2.3.0.0)

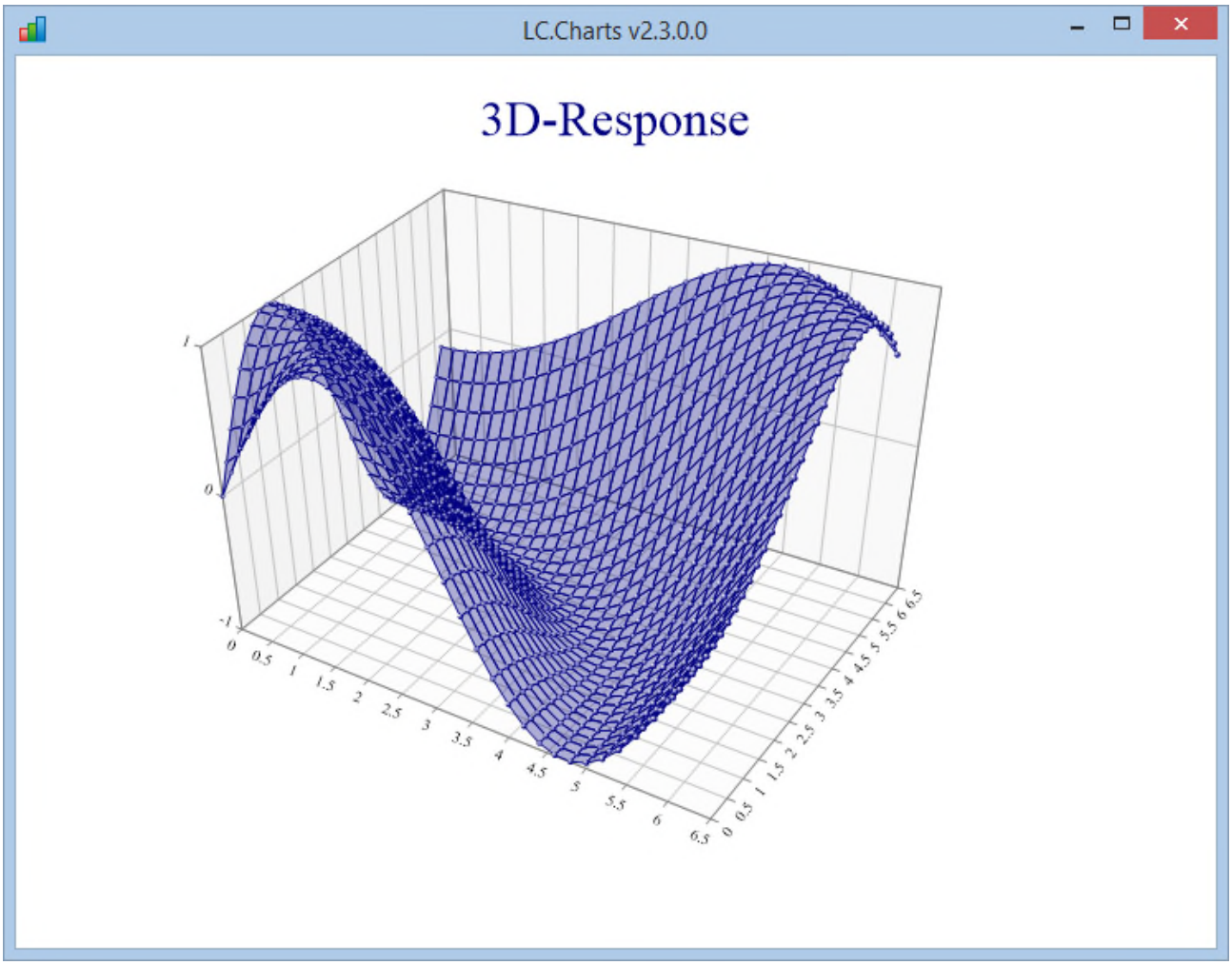
It is now possible to display the true X and Y values in Response plots instead of the point numbers along the X and Y axis. To do so, just provide the X and Y values in addition to the Z values.

For example, the **Sample3DResponse** function that produces the above chart reads as follows:

```
    ∇ Sample3DResponse;a;y;x;z;f
[1]   y←x←0(2×○1)Step.2
[2]   a←⎕def⊃[⎕io+1]'z←x f y' 'z←1○(((x*2)+y*2)*.5)'
[3]   z←⎕split x∘.f y
[4]   ⎕ucmd']chart3D z /ti=3D-Response /zmaj=1 /to /re /ty=response'
    ∇
```

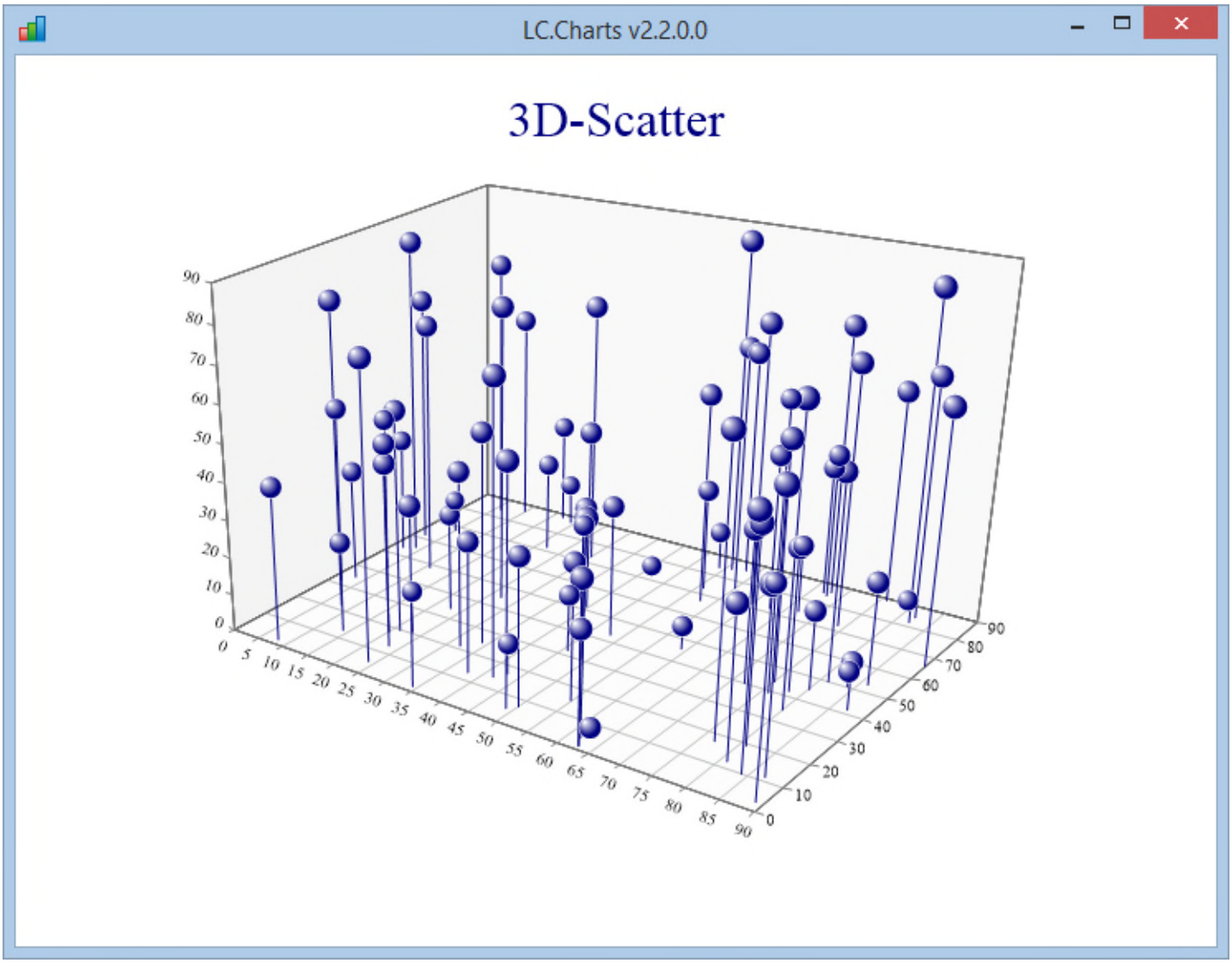
Simply change it to the following, to display the real X and Y values:

```
    ∇ Sample3DResponse;a;y;x;z;f
[1]   y←x←0(2×○1)Step.2
[2]   a←⎕def⊃[⎕io+1]'z←x f y' 'z←1○(((x*2)+y*2)*.5)'
[3]   z←⎕split x∘.f y
[4]   ⎕ucmd']chart3D z x y /ti=3D-Response /zmaj=1 /to /re /ty=response'
    ∇
```



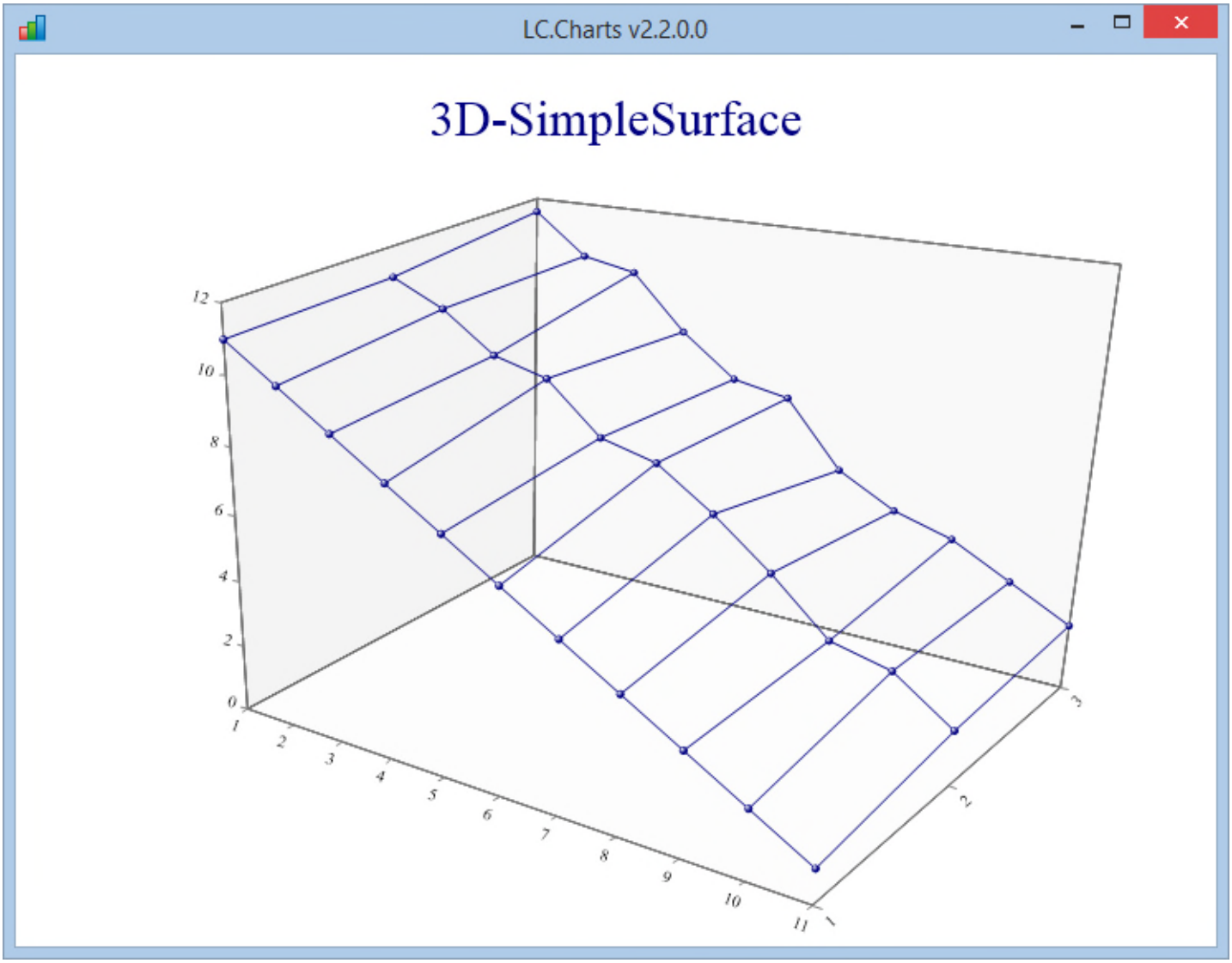
Scatter Plot Charts (/type=scatter)

This chart need no comment!



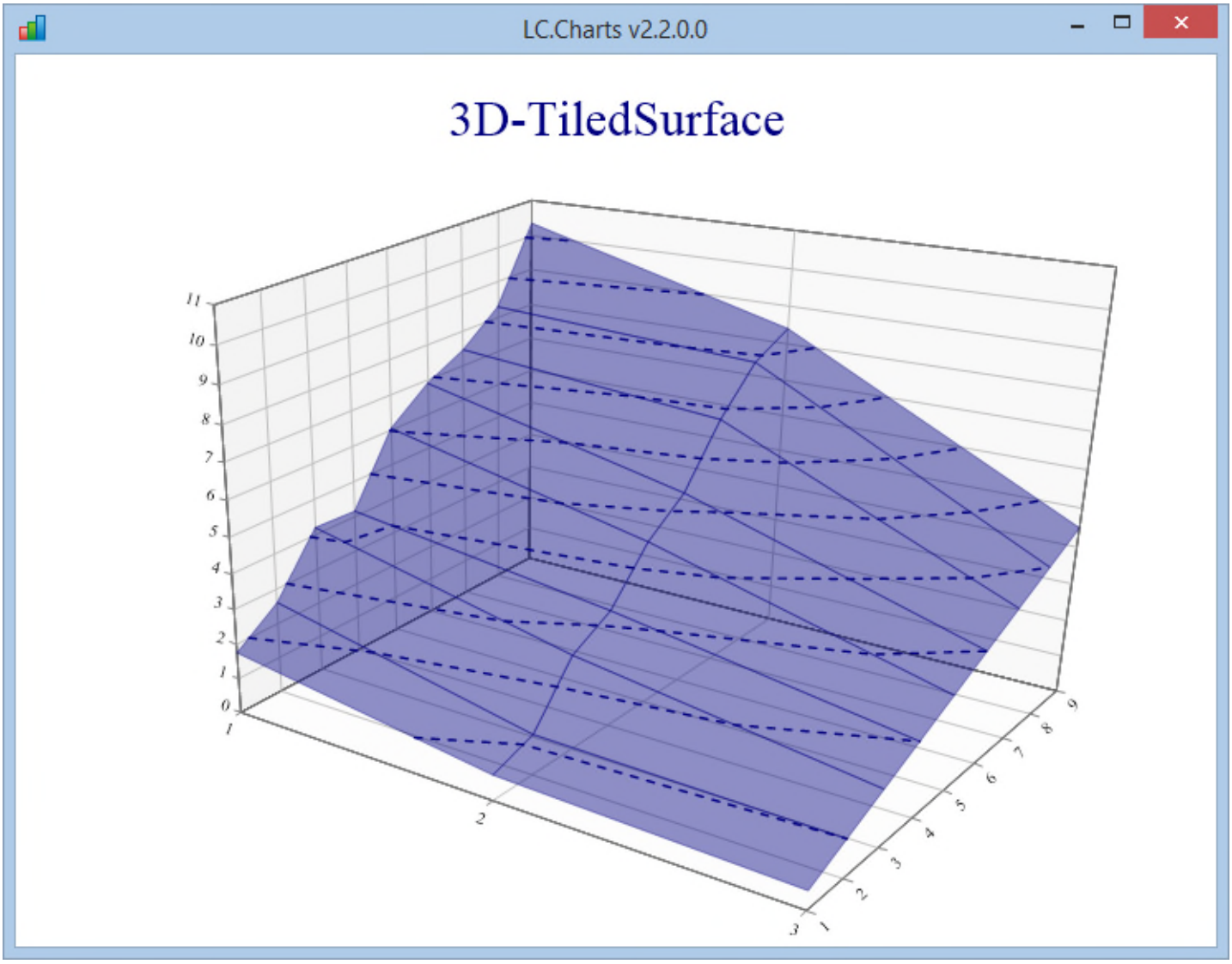
Simple Surface Charts (/type=simplesurface)

For this kind of surface, you must provide a number of Z values series. The implicit Y values will be the series numbers and the implicit X values the series point numbers.



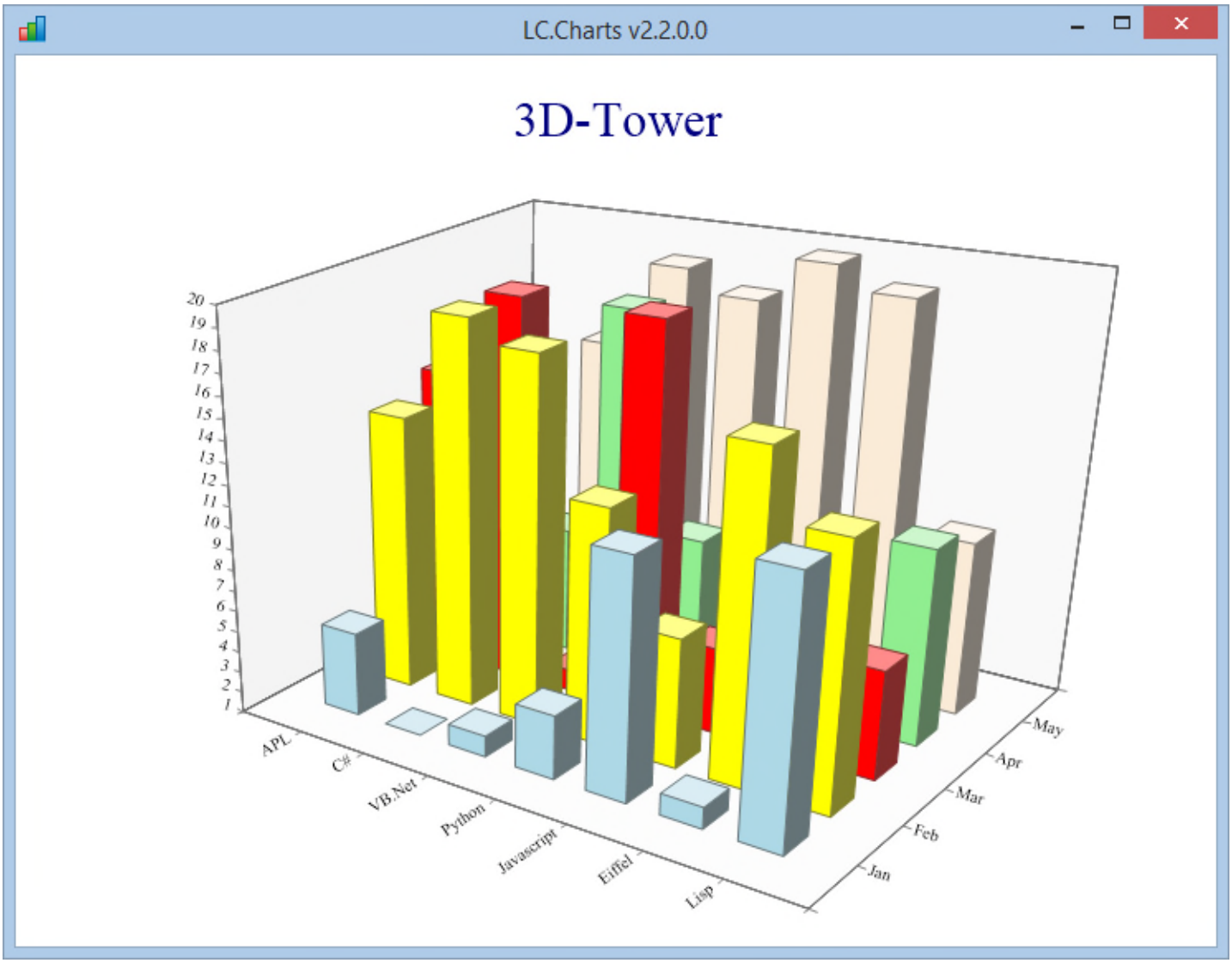
Tiled Surface Charts (/type=tiledsurface)

This type of chart is similar to the **simplesurface** chart with the difference that is uses a semi-transparent fill color to paint the surface and with the fact that it draws level curves.



Tower Charts (/type=tower)

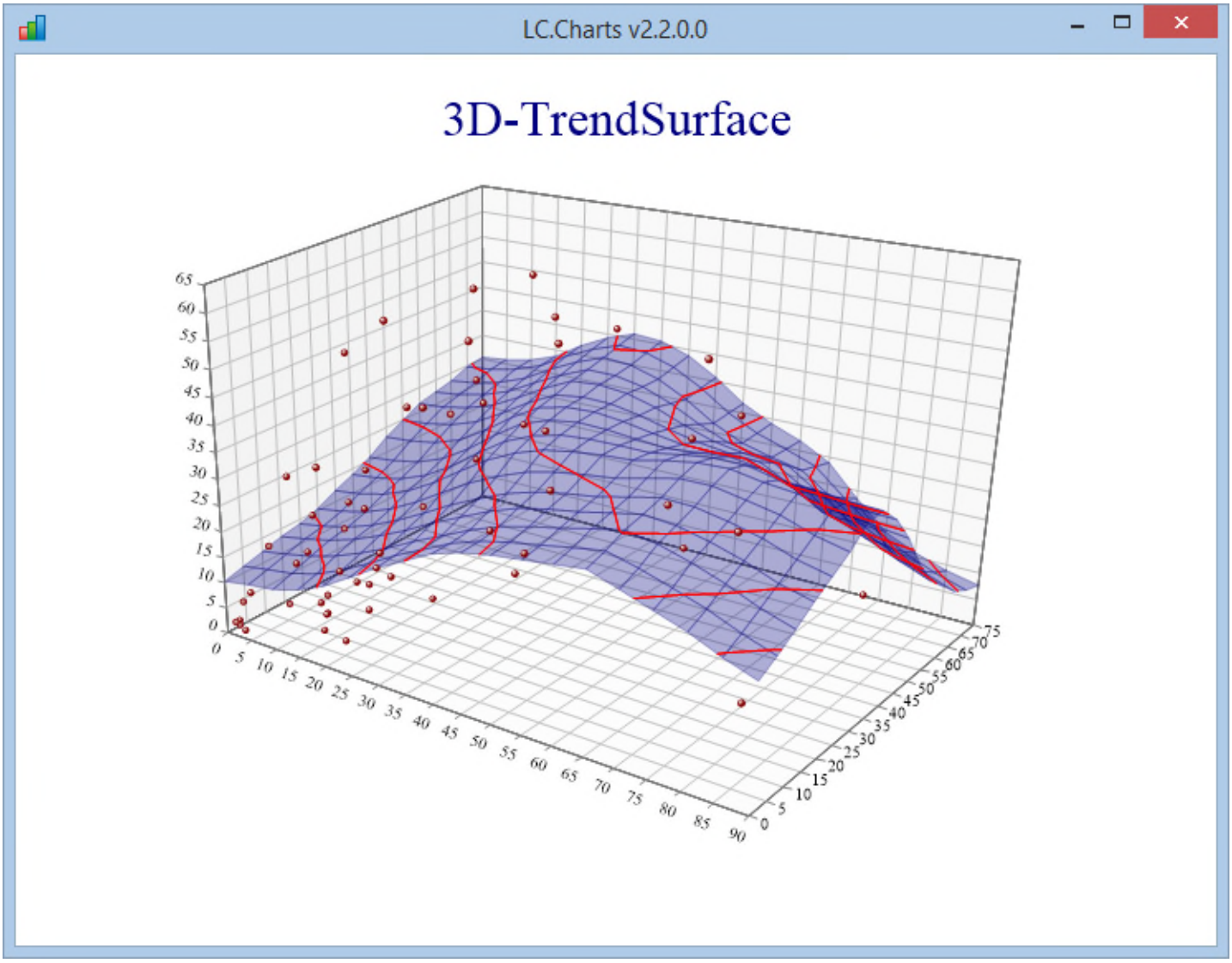
This well-known type of chart allows to represent a number of series, here software sales.



TrendSurface Charts (/type=trendsurface)

The trendsurface chart is a scatter plot with a gaussian-weighted smooth surface best fitting the points.

Curve level lines are drawn at each Z-axis tickmark as can be clearly seen when these lines cut the left hand side wall.



Customizing the 3D charts (new in v2.4.0.0)

You can use a number of wi properties and methods, new in v2.4.0.0, to customize any 3D chart you produce.

The margins (new in v2.4.0.0)

You could already change the top, right, bottom and left margins separately using the **xTopMargin**, **xRightMargin**, **xBottomMargin** and **xLeftMargin** properties.

You can now do the same with one instruction using the **XSetMargins** method (speficying margins in pixels, in the following order: top, right, bottom, left):

```
'ff.cc'wi'*XSetMargins'20 0 20 20
```

The default margins for all 3D charts are now set to: **50 0 40 56** except for the FlatTower charts where the default is: **50 30 40 30**.

The titles (new in v2.4.0.0)

You can set the 3D chart title using the **xTitle** property.

You can set the 3D chart subtitle using the **xSubTitle** property.

You cans et the X, Y and Z-Axis titles using the **xXAxisTitle**, **xYAxisTitle** and **xZAxisTitle** properties.

The 3D chart styles (new in v2.4.0.0)

You can now fully change the 3D chart styles for any of the 10 types of 3D charts.

To make it simple, the style property is named after the 3D chart type property with a suffix of **Style**.

So, for example, the style property for the xTrendSurface type of chart is called **xTrendSurfaceStyle** and the style property for the xTower type of chart is called **xTowerStyle**.

So the new Style properties are:

- **xAltitudeStyle**
- **xFitSurfaceStyle**
- **xFlatTowerStyle**
- **xPlanesStyle**
- **xResponseStyle**
- **xScatterStyle**
- **xSimpleSurfaceStyle**
- **xTiledSurfaceStyle**
- **xTowerStyle**
- **xTrendSurfaceStyle**

To easily find out what are the possible values you can use for any of these Style property, there is an associated property with the same name, but plural. The Styles properties therefore are:

- **xAltitudeStyles**[23]
- **xFitSurfaceStyles**
- **xFlatTowerStyles**
- **xPlanesStyles**
- **xResponseStyles**
- **xScatterStyles**
- **xSimpleSurfaceStyles**
- **xTiledSurfaceStyles**
- **xTowerStyles**
- **xTrendSurfaceStyles**

So, for example, if you want to now the styles you can use for an **xFitSurface** type of chart, you can do:

```
      'lccharts'⎕wi'*Create' 'LC.Charts.Chart3D'
lccharts
      ⎕wi'*xFitSurfaceStyles'
 AltitudeShading Contours Curves Fine FlatText GridLines HaloMarkers
       Indexed Lines ModelFit NoAxes NoMarkers ProjectContours Risers
       ScaleBar SurfaceShading TiledSurface TrendSurface WallShading
       XYPlot
```

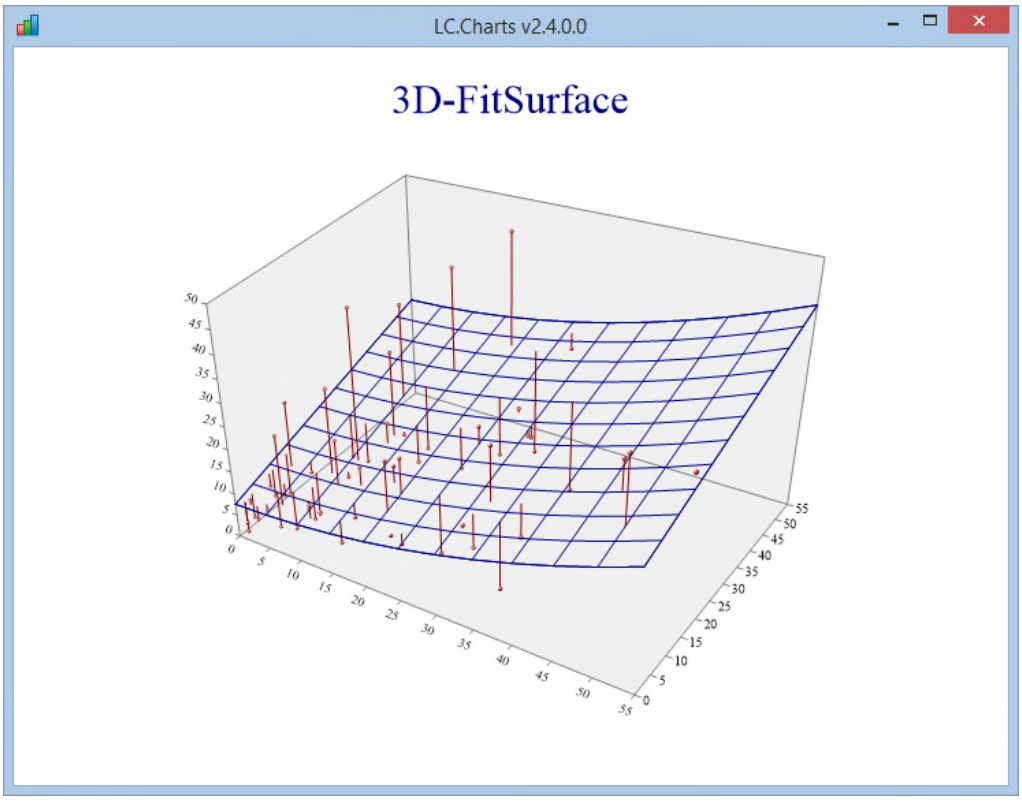
To apply a number of different styles to a 3D chart, just separate them with a comma.

Please not that the styles are case sensitive.

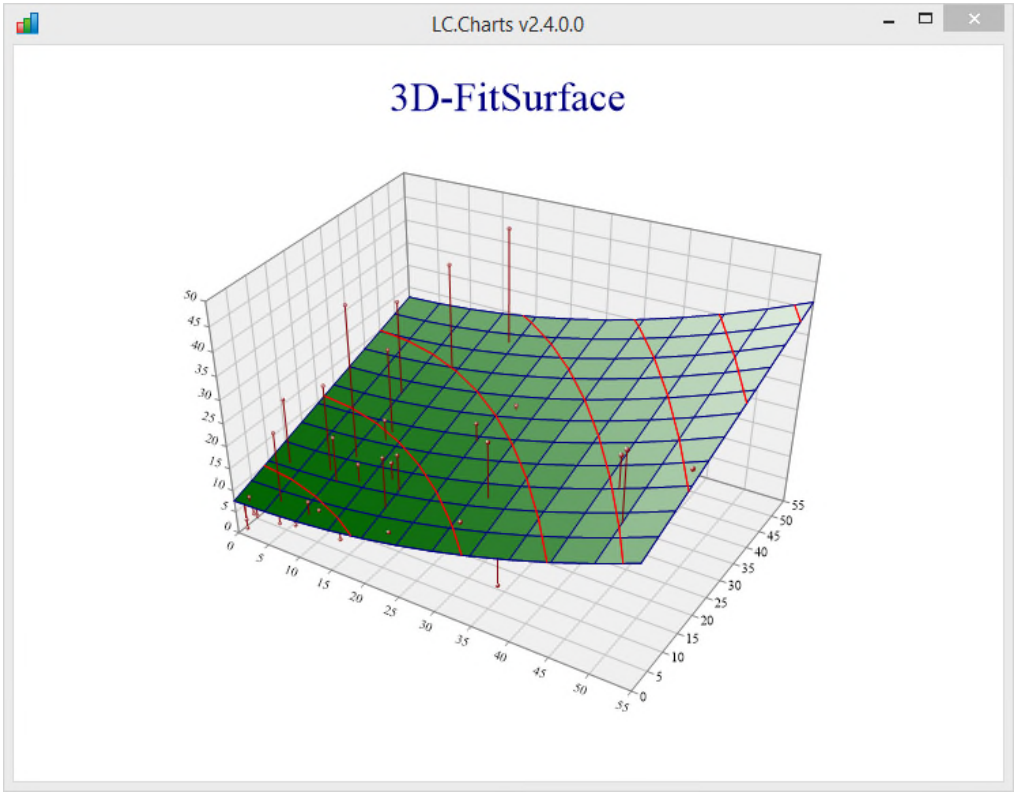
Example:

Sample3DFitSurface

[23] You may notice that some of the x…Styles properties return the same set of possible styles value. This may seem redundant, but I found it simpler and clearer to define an x…Style and an x…Styles property for each type of chart. And since they are named after the chart type, it is much easier this way to remember the name of the x…Style and x…Styles property to use.



wi'*xFitSurfaceStyle' 'AltitudeShading,Contours,GridLines,ModelFit,WallShading'



The best and most simple way to explore the effect of the various possible 3D chart styles is to use one of the sample 3D chart APL functions (Sample3DAltitude, Sample3DFitSurface, …, Sample3DTrendSurface) to display a 3D chart, then to query the corresponding Styles property (xAltitudeStyles, xFitSurfaceStyles, … or xTrendSurfaceStyles) to know which styles you can apply and finally to use the Style property (xAltitudeStyle, xFitSurfaceStyle, … or xTrendSurfaceStyle) to set some styles and immediately see the effect on the chart!

To make it easy, each of the predefined 3D chart has a default set of styles applied to it: the defaults are the following:

Altitude	`WallShading,TiledSurface,AltitudeShading`
FitSurface	`WallShading,ModelFit,Risers`
FlatTower	`WallShading,GridLines`
Planes	`NoMarkers,SurfaceShading,Lines,GridLines`
Response	`WallShading,GridLines,Markers,TiledSurface`
Scatter	`WallShading,Risers,HaloMarkers`
SimpleSurface	`WallShading,Markers`
TiledSurface	`WallShading,GridLines,TiledSurface,Contours`
Tower	`WallShading`
TrendSurface	`WallShading,TrendSurface,Contours,GridLines,TiledSurface`

You can change these defaults at any time, but the styles are not cumulative to the defaults, so for example, if you just want to add a **GridLines** style to a **Scatter** plot, you should not just use:

```
wi'*xScatterStyle' 'GridLines'
```

but rather:

```
wi'*xScatterStyle' 'WallShading,Risers,HaloMarkers,GridLines'
```

The X-, Y- and Z-Axis styles (new in v2.4.0.0)

You can now change the styles of any of the X-, Y- and Z-Axis.

Similarly to the 3d chart Style and Styles properties described above, there are:

- **xXAxisStyle**, **xYAxisStyle** and **xZAxisStyle** properties
- **xXAxisStyles**, **xYAxisStyles** and **xZAxisStyles** properties

The possible (not default!) values are for the styles are:

xXAxisStyles	`AngledLabels AnnualTicks ArrowedAxis AtEndCaption BalancedAxis CenteredCaption Clipped Date DuplicateAxes Duration ExactFit FlatText ForceZero GridLines InsideLabels InvisibleAxis LogScale Longitude MaskedLabels MiddleLabels MonthlyTicks NoAxis NoLabels OmitLastLabel OverlayGrid PlainAxis RightCaption SpannedLabels StayAtEdge Time`

	TopAxis InsideLabels ExactFit
xYAxisStyles	AboveGrid AngledLabels ArrowedAxis AtEndCaption BalancedAxis CenteredCaption ClampToAxis Clipped Date DuplicateAxes Duration ExactFit FlatText ForceZero GridLines InsideLabels InvertAxis InvisibleAxis Latitude LeftAxis LogScale MiddleLabels NoAxis NoLabels OverlayGrid PlainAxis RightAxis StayAtEdge Synchronised Time
xZAxisStyles	AtEndCaption ExactFit FlatText ForceZero GridLines NoLabels PlainAxis

As for 3D chart styles:

- the styles are not cumulative to the default styles
- the styles are case sensitive
- you should separate the styles you want to apply with a comma

And as for the 3D chart styles, the best way to explore these axis styles is to draw a 3D chart topmost (for example using one of the supplied Sample3Dxxxxx functions), set various X-Axis, Y-Axis and/or Z-Axis styles from the APL Session and observe the immediate effects on the chart.

Axis Labels Formats (new in v2.4.0.0)

You can use the 3 new **xXAxisFormat**, **xYAxisFormat** and **xZAxisFormat** properties to change the labels format along the X-, Y- and Z-Axis.

These properties work like the same named properties for 2D charts.

The values you provide to these properties must conform to the .Net format strings (namely the DateTimeFormatInfo and NumberFormatInfo classes possibilities).

You can find a good cheat sheet summaraizing all the .Net formatting capabilities at:

http://www.cheat-sheets.org/saved-copy/msnet-formatting-strings.pdf

See the examples provided in the **Customizing the axis labels (new in v2.3.0.0)** paragraph.

Note that in the case of 3D charts there are a coupld of exceptions: the '%' symbol has no effect on the scale of the data, and you may use the '~' character anywhere in the picture to suppress the digit at that position.

In the case of a 3D chart you can also specify a format for text labels: is you specify an x_AxisFormat of XXXXXXXX for text labels, any labels with text longer than XXXXXXXX will be truncated with … ellipsis appended to the label.

Adding notes to a 3D chart (new in v2.4.0.0)

You can add a note anywhere on a 3D chart at any time using the AddNote method.

Its syntax is:

```
'object'□wi'AddNote' note x y angle width style font backColor
```

where:

note	Is the text of your note. This text will be automatically wrapped by default in the width you specify in the 5th argument
x	The x position (possibly negative) in the chart window at which you want to display the note The origin is at the bottom left and the x position must be expressed in % of the chart width
y	The y position (possibly negative) in the chart window at which you want to display the note The origin is at the bottom left and the y position must be expressed in % of the chart height
angle	The angle to apply to the note Please note that the style (6th argument) is ignored unless the angle is 0.
width	The width in pixels in which you want to display the note
style	A comma separated set of styles (see the **xNoteStyles**[24] property)
font	A nested vector: fontName fontSize fontStyle fontColor Only the fontName is compulsory, so you could define the font as: 'Impact' 'Impact'7 'Impact'7 2 'Impact'7 2(255 0 0) The possible font styles are: 0=regular, 1=bold, 2=italic, 4=underline or any combination of those by adding values (example: 3=bold+italic) The fontColor must be expressed as a 3-element RGB vector (example: 0 0 255)
backColor	A nested vector: color fill where : color is a 3-element RGB color value (example: 192 192 0) fill is one (and only one) of the possible fill styles returned by the **xFillStyles**[25] property

Here is how you can easily create a note:

---

[24] The possible note styles are:

□wi'*xNoteStyles'

Absolute Baseline Bottom Boxed CenterAlign Cropped Middle Opaque Percentage RightAlign Rounded Shadowed TopAlign

[25] The possible fill styles are:

□wi'*xFillStyles'

BackwardDiagonal DiagonalCross Dome Drum ForwardDiagonal GradientBottom GradientBottomLeft GradientBottomRight GradientLeft GradientTop GradientTopLeft GradientTopRight Halftone HorizontalBrick Opacity18 Opacity30 Opacity42 Opacity54 Opacity6 Opacity66 Opacity78 Opacity90 Pipe Saturate Saturate10 Saturate20 Saturate30 Saturate60 Saturate80 Saturate90 Shingle Solid SpotBottomLeft SpotBottomRight SpotTopLeft SpotTopRight Sunrise Unfilled Weave

```
note←'This chart shows a clear relationship between the evolution of
Mortality under 5 and the GDP'
fill←(192 255 255)'GradientBottomRight'
font←'Arial'7 2(255 0 0)
style←'Boxed,Shadowed,Percentage'
⎕wi'*XAddNote'note ¯10 14 0 60 style font fill
```

You can run the **Demo24** sample function to see this example in effect.

Adding a footnote to a 3D chart (new in v2.4.0.0)

You can also add a footnote to any 3D chart, using the **XAddFootnote** method.

Its syntax is:

'object'⎕wi'AddFootote' footnote style font

where:

note	Is the text of your note. This text will be automatically wrapped by default in the width you specify in the 5th argument
style	A comma separated set of styles (see the **xFootnoteStyles**[26] property)
font	A nested vector: fontName fontSize fontStyle fontColor Only the fontName is compulsory, so you could define the font as: 'Impact' 'Impact'7 'Impact'7 2 'Impact'7 2(255 0 0) The possible font styles are: 0=regular, 1=bold, 2=italic, 4=underline or any combination of those by adding values (example: 3=bold+italic) The fontColor must be expressed as a 3-element RGB vector (example: 0 0 255)

Here is an example of creating a footnote:

```
note←'Source: http://data.worldbank.org'
font←'Arial'7 0(3ρ160)
style←'Right'
z←⎕wi'*XAddFootnote'note style font
```

[26] The possible footnote styles are:
⎕wi'*xFootnoteStyles'
Center NoWrap Right RuledAbove

You can run the **Demo24** sample function to see this example in effect.

Changing the perspective (new in v2.4.0.0)

You can now change the perspective from which the 3D chart is observed using the new **xPerspective** property.

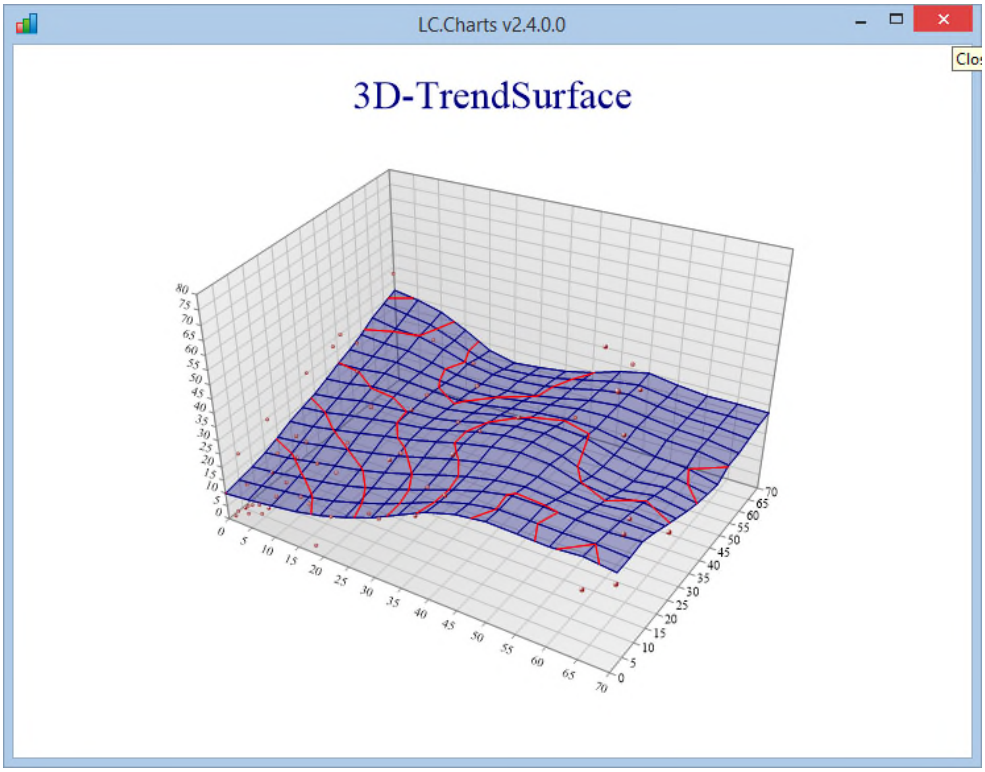
The default xPerspective value is: 12

If you set the xPerspective property to a value less than 12 you'll see the chart from a closer point, i.e. increasing the perspective effect.

If you set the xPerspective property to a value greater than 12, y'oull see the chart from further away finally resulting for an xPerspective value fo 50 or above to parallel vertical axis.

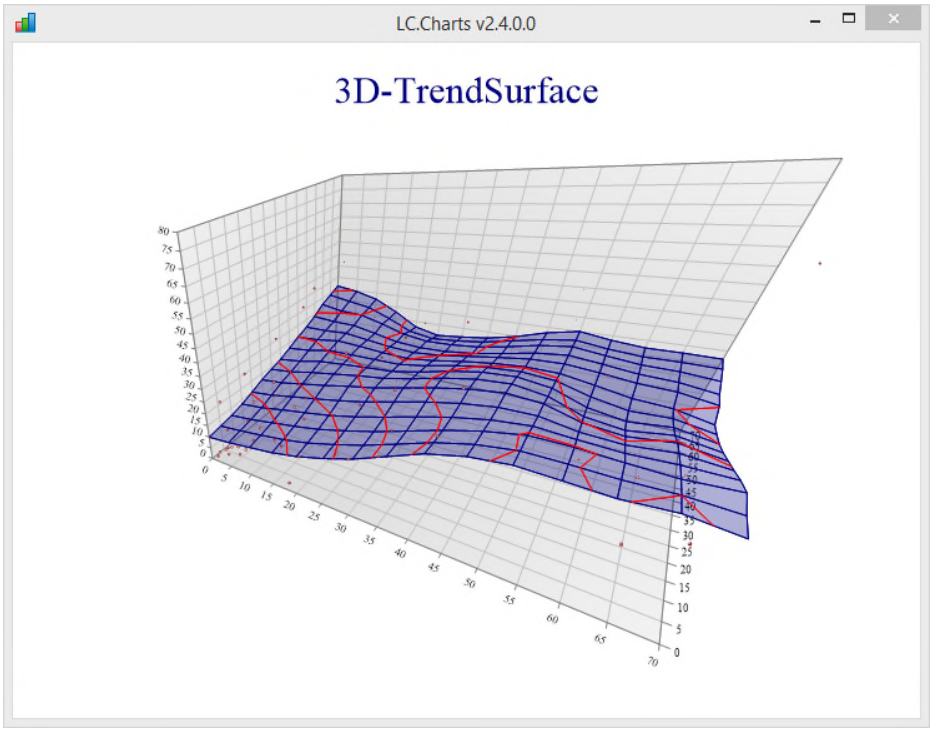
Example:

```
Sample3DTrendSurface
```

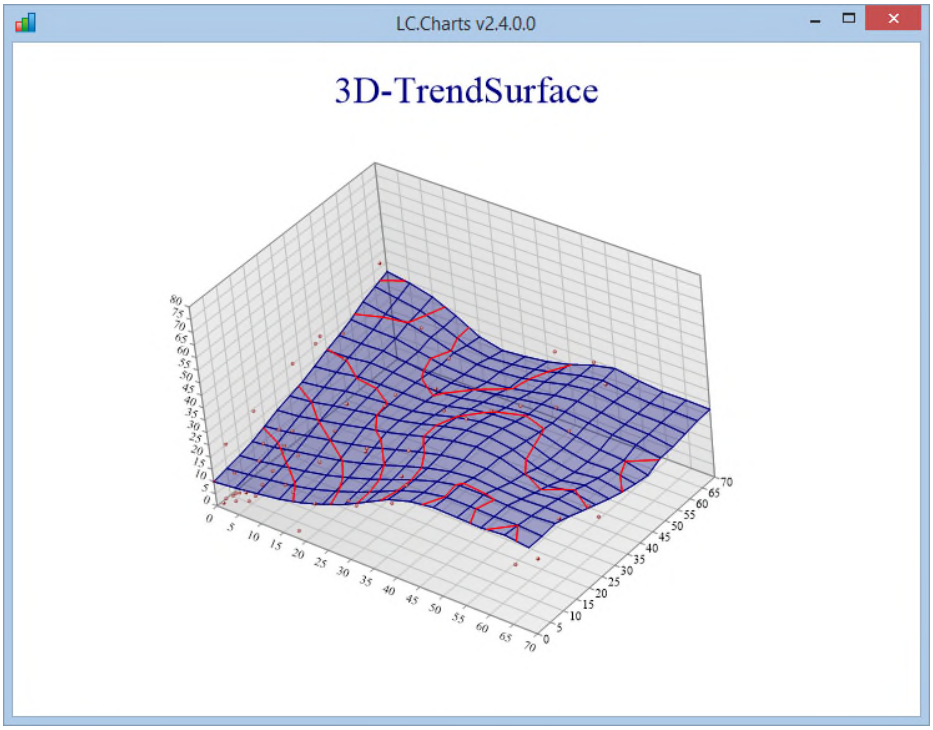



```
wi'*xPerspective'
12
```

```
wi'*xPerspective' 5
```



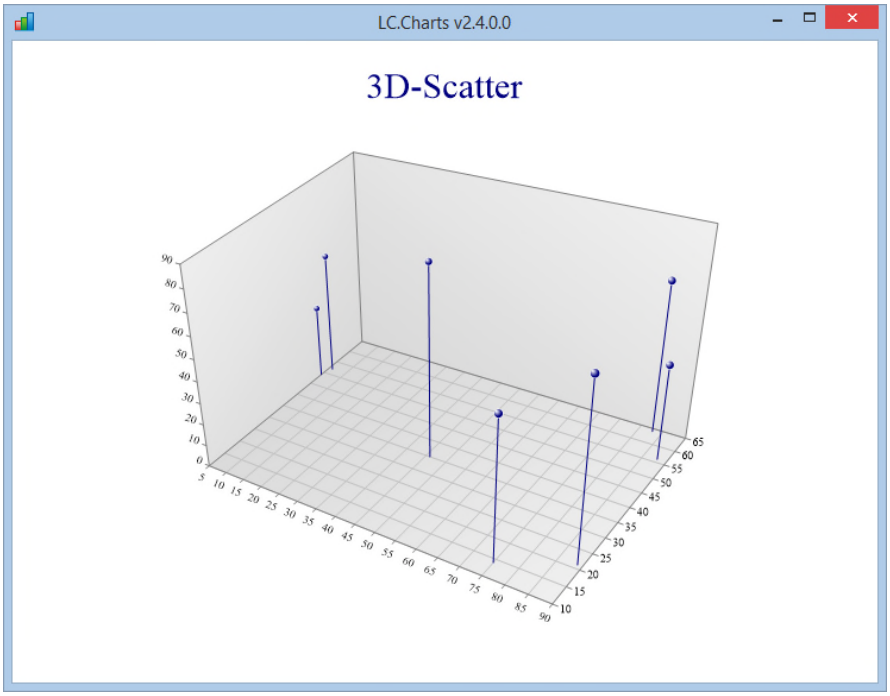

```
wi'*xPerspective' 50
```



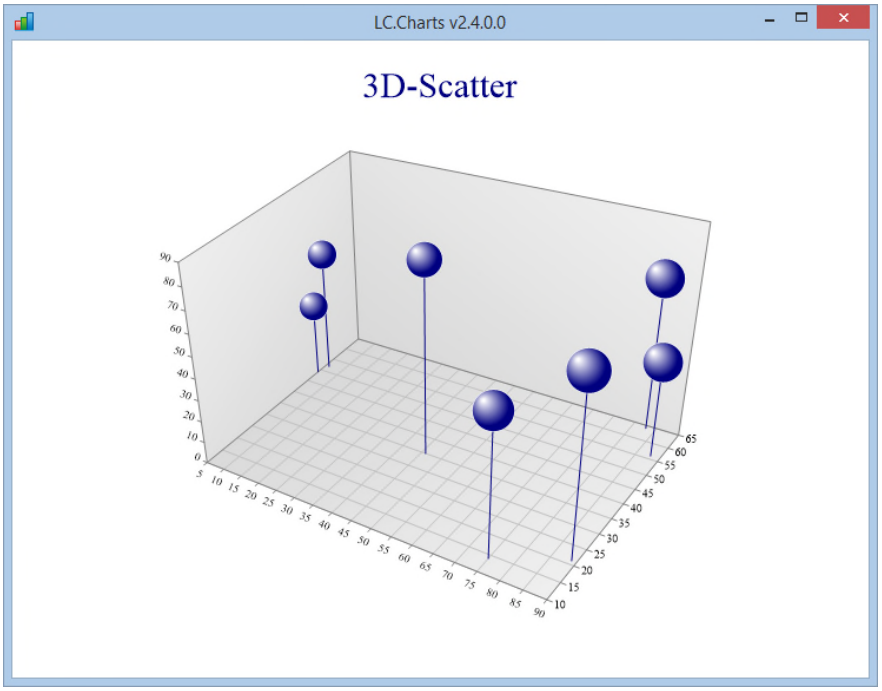
Changing the symbol/marker size (new in v2.4.0.0)

You can now change the marker size for 3D Scatter plots and other 3D plots using the **xSymbolSize** property.

```
Sample3DScatter
```




```
wi'*xSymbolSize'5
```



The Demo24 APL function (new in v2.4.0.0)

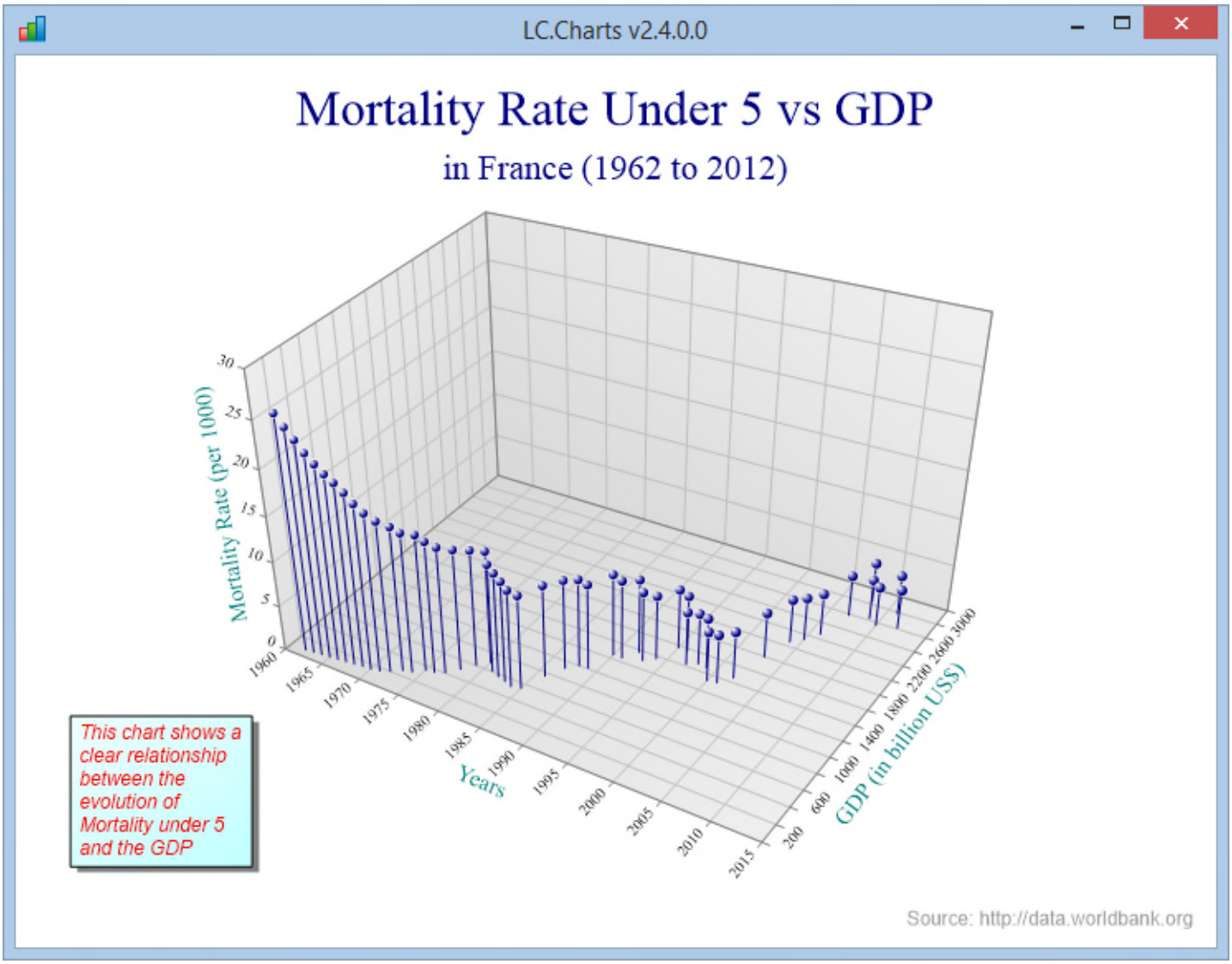
The new Demo24 APL function delivered in the LCCHARTS.SF User Command file demonstrates all the new properties and methods in included in LC.Charts v2.4.0.0.

Here it is:

```
    ∇ Demo24;data;fill;font;note;style;z
[1]   ⍝∇ Demo24 -- This function demonstrates the new properties available in v2.4.0.0
[2]   ⍝∇ (c) 2015 Lescasse Consulting
[3]
[4]   ⍝ Get some data
[5]   data←1 1 0 1/⍳,/11 10 WorldDataEvolution¨8
[6]
[7]   ⍝ Draw some 3D chart
[8]   ⎕ucmd']chart3D ⎕split⍉1 0↓data /ti=3D-Scatter /to /re /ty=scatter'
[9]   z←⎕wi'*xTopMargin'50
[10]  z←⎕wi'*xBottomMargin'35
[11]  z←⎕wi'*xRightMargin'0
[12]  z←⎕wi'*xLeftMargin'56
[13]  z←⎕wi'*xZAxisStyle' 'ForceZero,GridLines'
[14]  z←⎕wi'*xXAxisStyle' 'GridLines,AngledLabels'
[15]  z←⎕wi'*xYAxisStyle' 'GridLines'
[16]  z←⎕wi'*xTitle' 'Mortality Rate Under 5 vs GDP'
[17]  z←⎕wi'*xSubTitle' 'in France (1962 to 2012)'
[18]  z←⎕wi'*xZAxisTitle' 'Mortality Rate (per 1000)'
[19]  z←⎕wi'*xXAxisTitle' 'Years'
[20]  z←⎕wi'*xYAxisTitle' 'GDP (in billion US$)'
[21]  z←⎕wi'*xYAxisFormat' '0,,,'  ⍝ divide values by 10*9
[22]  z←⎕wi'*xScatterStyle' 'WallShading,Risers,HaloMarkers'
[23]  note←'This chart shows a clear relationship between the evolution of Mortality
under 5 and the GDP'
[24]  fill←(192 255 255)'GradientBottomRight'
[25]  font←'Arial'7 2(255 0 0)
[26]  style←'Boxed,Shadowed,Percentage'
[27]  z←⎕wi'*XAddNote'note ¯10 14 0 60 style font fill
[28]  note←'Source: http://data.worldbank.org'
[29]  font←'Arial'7 0(3○160)
[30]  style←'Right'
[31]  z←⎕wi'*XAddFootnote'note style font
    ∇
```

Running this function produces the foolowing chart:

Demo24



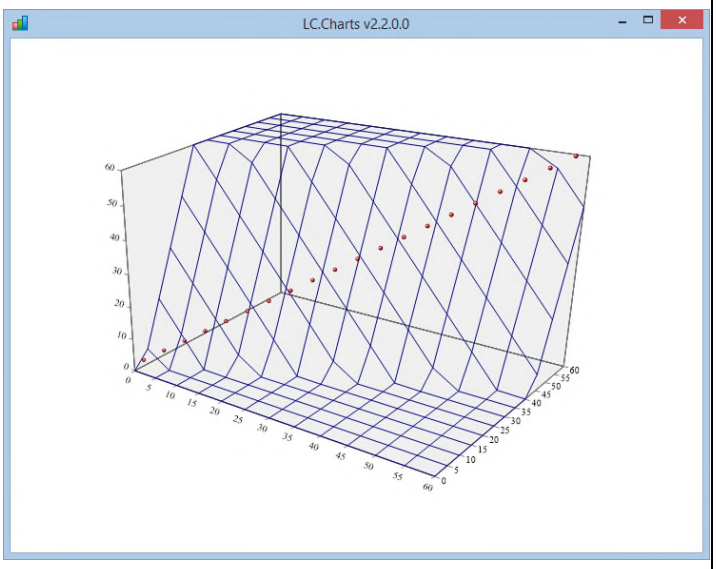
Spinning the 3D charts (new in v2.2.0.0)

One of the first things you want to do with a 3D chart is to change the perspective or “viewpoint” from which you look at it.

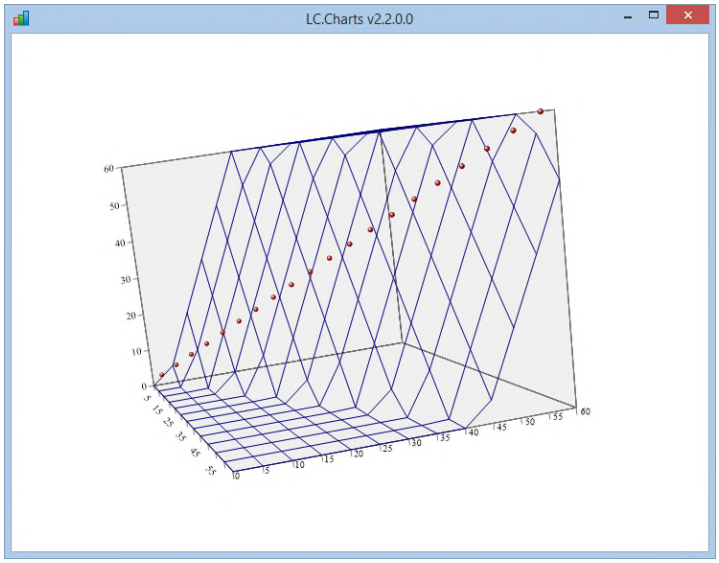
The xViewPoint property (new in v2.2.0.0)

You can use the **xViewPoint** property to change the viewpoint. Example:

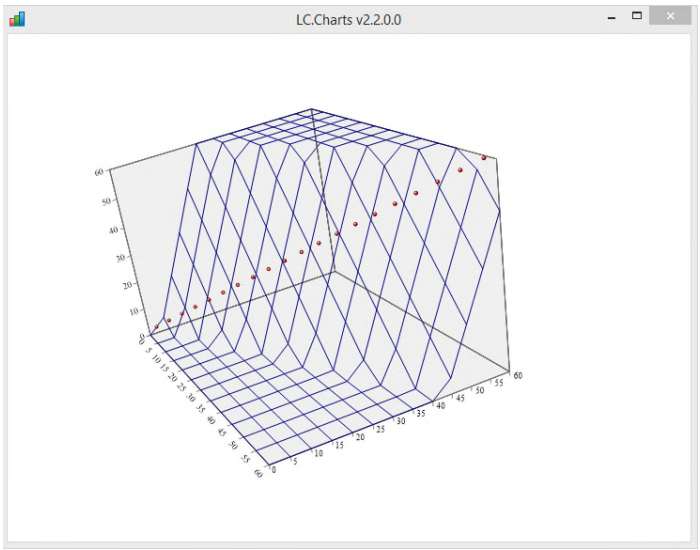
```
]chart3d □split◊20 3ρι100 /to /re
```



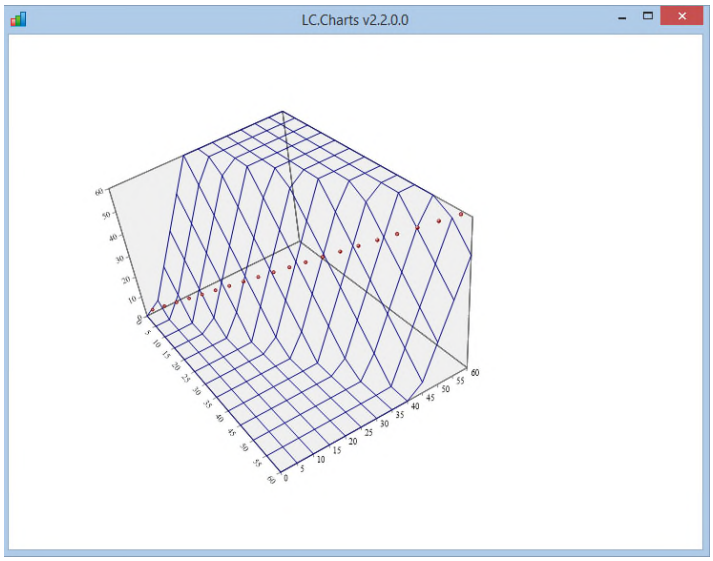

```
□wi'*xViewPoint'10 10 10
```

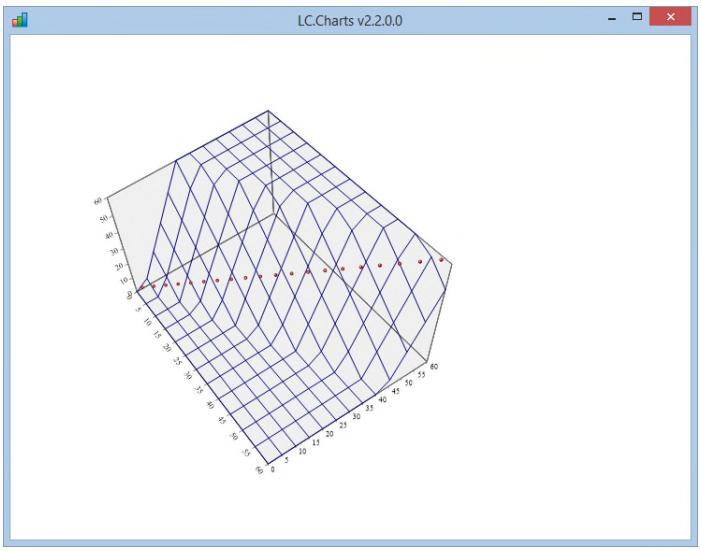
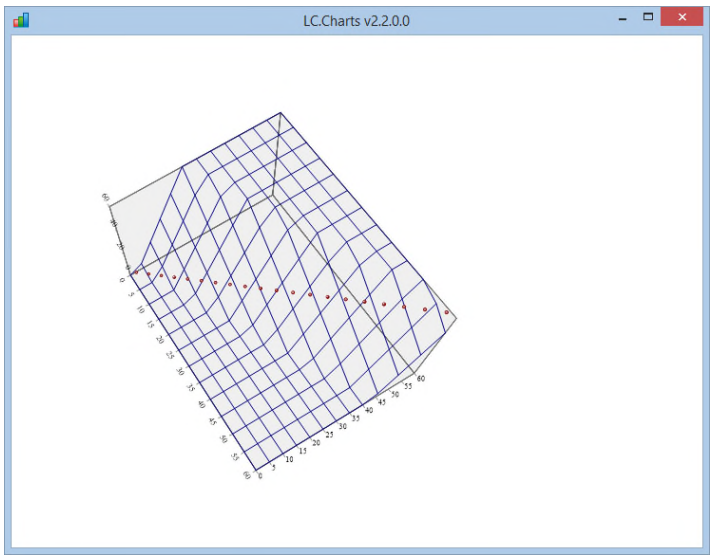
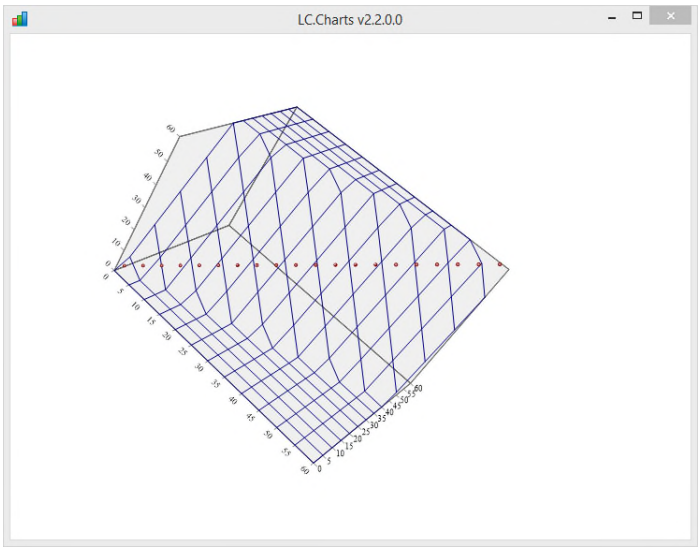
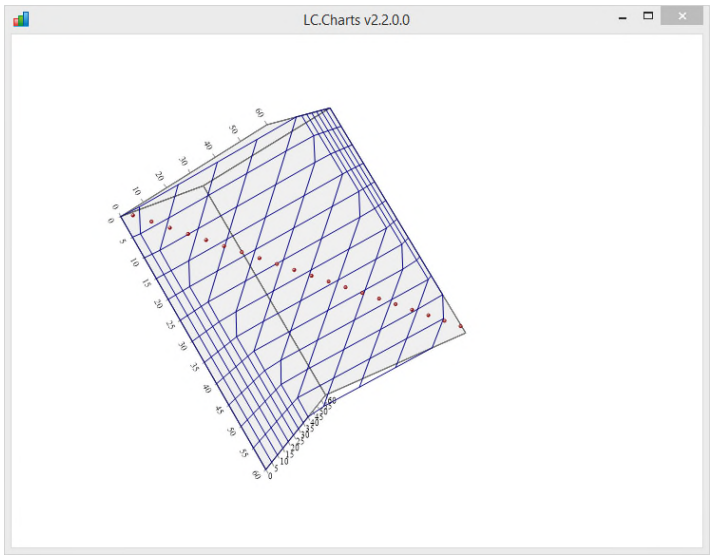



```
□wi'*xViewPoint'20 20 20
```




```
□wi'*xViewPoint'30 30 30
```



`wi'*xViewPoint'40 40 40` 	`wi'*xViewPoint'50 50 50` 
`wi'*xViewPoint'20 40 60` 	`wi'*xViewPoint'20 60 80` 
Etc.	

There is no clear rule concerning the 3 values you must apply: you'll really have to experiment with **xViewPoint** to find the best angle to look at your data.

The]spinit user command (new in v2.2.0.0)

Once you have created a 3D chart with **]chart3D**, you can make it spin so that you can inspect your 3D chart from all possible angles.

Just create your chart using **]chart3d** (which all **Sample3Dxxxxx** functions do) or with ⎕wi[27]

Then run the **]spinit** user command with no argument.

You may optionally provide an argument to **]spinit** which should be an integer between **10** and **2000**.

The higher the value you provide to **]spinit**, the slower the 3D chart will spin, but at the same time the more smoothly it will spin and the better you'll be able to inspect your chart from all angles.

If you don't provide a value, the default value will be **360**.

Before using **]spinit**, you must know that some charts are quite slower to draw than others: in general the surfaces are slower (as the number of points is generally much higher than in a Tower chart for example).

So it is advidable to choose values below **100** for the altitude shading plots for example.

Try:

```
Sample3DTower

]spinit

]spinit 10

]spinit 720
```

Manually rotating 3D charts with your mouse (new in v2.3.0.0)

Starting with v2.3.0.0, it is now possible to drag your mouse on the chart surface to manually rotate any 3D chart.

However, due to the way observation points are implemented in the underlying graphic package that is used, it is very difficult to achieve pure horizontal or pure vertical rotations.

Consequently you'll have to get used to use your mouse to rotate the chart.

One advice is to move your mouse pretty slowly in kind of circle movements.

And remember that, in order to rotate the 3D chart, you must keep your mouse button down while you move your mouse. As soon as you release the mouse button, mouse moves stop rotating the chart.

[27] But if you use ⎕wi be sure to name your 3D chart object instance: **lccharts3d** (in the case of **LC.Charts.Chart3D**) or **lccharts3d.cc** (in the case of **LC.Charts.Charts3DUC** when you embed the chart in an APL+Win form)

Out of Memory problems and Errors sometimes occurring while manually rotating 3D charts fixed (new in v2.5.0.0)

Manually rotating 3D charts with your mouse sometimes resulted in Out of Memory Exceptions or in some other errors being displayed in a MessageBox by LC.Charts. These problems have been fixed in v2.5.0.0.

Changing the 3D chart margins (new in v2.3.0.0)

You can change the 3D chart margins using the following 4 properties:

- **xBottomMargin**
- **xTopMargin**
- **xRightMargin**
- **xLeftMargin**

All margins should be expressed in pixels.

Support for accented letters and the € and £ symbols (new in v2.3.0.0)

Starting with v2.3.0.0, LC.Charts supports accented letters from all countries and also supports the € symbol and the £ symbol.

This has been done at the C# level[28] to prevent you from having to use the AV2ANSI and ANSI2AV APL utilities.

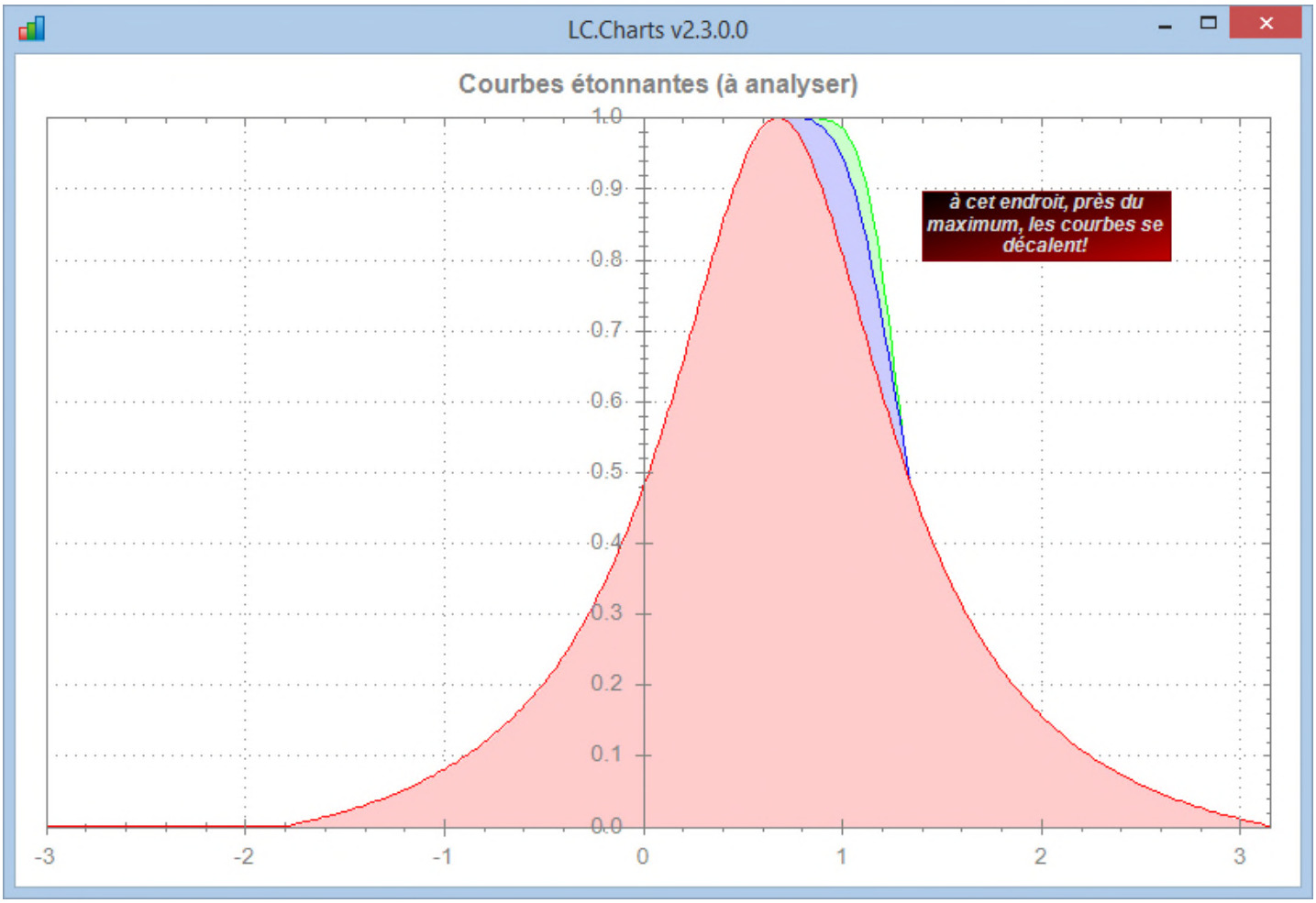
Therefore you can very naturally write:

```
      ]chart SampleData1 /cf=80 /to
      ⎕wi'*xTitle' 'Courbes étonnantes (à analyser)'
      ⎕wi'*xTitle'
Courbes étonnantes (à analyser)

      annot←'à cet endroit, près du'
      annot←annot,⎕tclf,'maximum, les courbes se'
      annot←annot,⎕tclf,'décalent!'
      textBorder←1(128 0 0)((0 0 0)(192 0 0)45)
      textFont←'Helvetica'12 3(224 224 224)
      ⎕wi'*xAddText'annot 1.4 0.8 textBorder textFont
      ⎕wi'*XRedraw'
```

Note that in the chart presented next page all accents are correctly rendered.

[28] In fact an AV2ANSI extension method and an ANSI2AV extension method have been created in the C# ActiveX to automatically convert accented letters and € and £ when you supply them from APL.



Conclusion

I hope this document will help you better see how to use LC.Charts.

In the future, I plan to extend LC.Charts by adding many more □wi properties, methods and possibly events.

However, I will probably not add more options to the **]chart** command so that it remains "manageable"!

Remember that this tool is primarily made for being used in the APL Session to explore your data and visualize them in charts, as easily as:

```
]chart myAplVariable
```

However, LC.Charts could also become a much more complete charting and graphing tool provided a lot more properties are made available through □wi.

It's nice and fun to develop a freeware, but it only makes sense if users have a real use for it and like it.

So please send me your feedback so that I can measure if what I am doing is useful or not, as well as your suggestions … and bug reports!

Appendix

Appendix 1: the APL+Win x and * property prefixes

The reason for the x prefix is simple and is explained in the APL+Win Manuals. Any object, even an ActiveX object which is not one of the standard APL+Win delivered object has a set of basic properties provided by APL+Win, like the **version** property for example. You can check that by doing:

```
]chart ?20⍴20

'lccharts'⎕wi'properties'
```

You'll see that there are a number of properties not prefixed by **x** and then all the ones prefixed by **x**. The first ones are the ones provided by APL+Win and basically common to all objects. The latter ones are the ActiveX object specific properties (i.e. the properties I have developed in LC.Charts.dll).

It is important to be able to distinguish between the **version** property (the one provided by APL+Win) and the **Version** property (the one provided by DLL). So APL2000 early decided that all properties coming from an ActiveX should be prefixed with a lowercase **x**.

So you can try:

```
'lccharts'⎕wi'*version'

'lccharts'⎕wi'*xversion'
2.0.0.0
```

It turns out that the APL+Win **version** property does not return anything in that case, but it could have: in any case, the 2 results are not the same.

The uppercase **X** prefix is reserved for methods.

Something else may be confusing and need be understood.

In APL+Win, for all standard APL+Win objects, properties, methods and event names are case sensitive.

However, in the case of ActiveX, properties, methods and events are NOT case sensitive.

So you can write:

```
      'lccharts'⎕wi'*xsymbol' 'square
      'lccharts'⎕wi'*xSYMBOL' 'square'
```

It is also allowed to omit the **x** prefix:

```
      'lccharts'⎕wi'*symbol' 'square'
      'lccharts'⎕wi'*SYMBOL' 'square'
```

However, in the case of ActiveX properties which naturally start with an X (like the X-Axis properties in LC.Charts), you MUST prefix the properties with an **x** because otherwise the system would think that the X in XAxisTitle would be the x prefix and not be part of the property name.

So you must write:

```
      'lccharts'⎕wi'*xXAxisType' 'Years'
```

This would not work:

```
      'lccharts'⎕wi'*XAxisType' 'Years'
⎕WI ACTION ERROR: Action "XAxisType" not found
      'lccharts'⎕wi'*XAxisType' 'Years'
                ^
```

In other words, you can omit the **x** prefix in the following cases (the majority of cases):

1. if there is no APL+Win standard property with the same name (i.e. version for example)
2. If the ActiveX property does not start with x

It may be a little bit confusing and that's why, to avoid any problem, my recommendation is to always:

1. prefix ActiveX properties with **x**
2. prefix ActiveX methods with **X**
3. use case sensitiveness when calling ActiveX properties or methods (i.e. use **xXAxisTitle** rather than **xxaxistitle** even though both would work).

Finally the * prefix is not required in most cases:

```
      'lccharts'⎕wi'symbol' 'square'
      'lccharts'⎕wi'SYMBOL' 'square'
      'lccharts'⎕wi'xXAxisType' 'Years'
```

It is only required when you want to call an APL+Win standard property that has the same name as an ActiveX property.

The * prefix basically means: call an APL+Win standard property (one which is not prefixed with an x in the list of properties):

```
'lccharts'⎕wi'*version'
```