SVG Charts in 4D v11 SQL

By Christopher Visaya, Technical Support Engineer, 4D Inc.

Technical Note 07-40

Abstract

Scalable Vector Graphics (SVG) is an XML-based language used to create rich, two-dimensional graphics. This file format has a few characteristics that allow for some advantages over more popular image types such as JPEG or GIF. For example, it is (as its name implies) scalable; SVG images are not pixel-based, they are vector-based images. They are built from geometric shapes, such as lines, which means they do not need to store information for each pixel.

4D v11 SQL features native support for SVG.

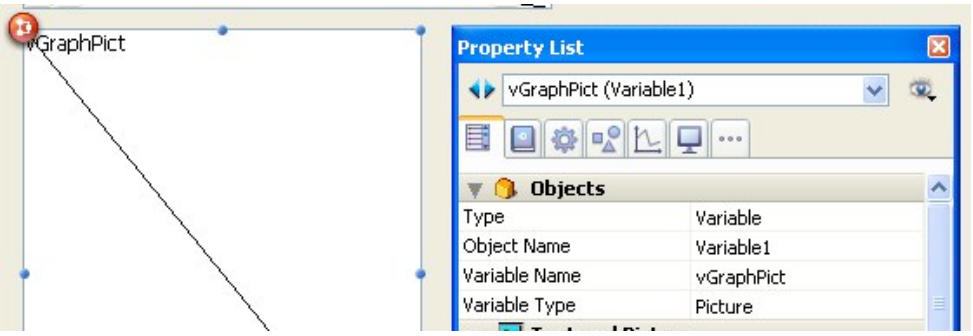
SVG is typically used to draw statistical data and, although the language is contained within an XML tree, 4D developers can modify SVG attributes directly within the 4D method editor. This document includes an example database to generate and modify statistical SVG graphs, output results in a form, and export the image as an SVG image to the disk.

Invoking SVG

4D v11 SQL has a built-in SVG rendering engine. There are a few ways within 4D to tap into this. One of those ways is with using the **GRAPH** command. This command has changed slightly from the past in that it is up to the user to decide which graphics engine to use based on the first parameter. The syntax for this command is as follows:

```
GRAPH(graphArea;graphNumber;xLabels;yElements{;yElements2;…;yElementsN})
```

The *graphArea* parameter determines which graphics engine will be used: passing a 4D Chart area or graph area reference will use the 4D Chart plug in. Passing a picture variable will make 4D use the SVG engine. In the sample database, observe that the input form has a variable of type Picture and thus whatever is graphed will be done using SVG:



Since the GRAPH command is being used, it is also necessary to use the **GRAPH SETTINGS** command. The syntax for this command is as follows:

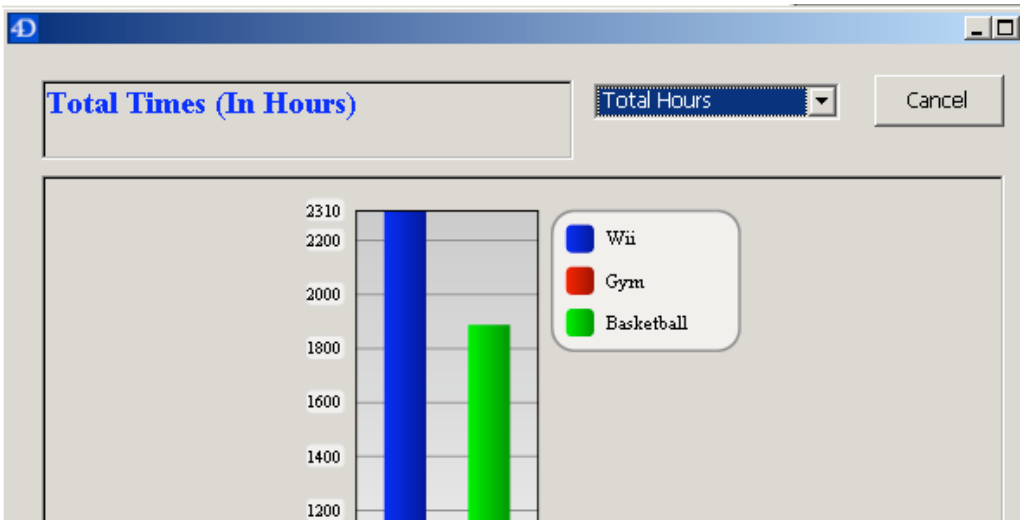
```
GRAPH SETTINGS(graph;xmin;xmax;ymin;ymax;xprop;xgrid;ygrid;title{;title2;…;titleN})
```

The *graph* parameter needs to match up with the *graphArea* parameter from the GRAPH command. Beyond that, setting the rest of the parameters is straightforward.

The Sample Database

The sample database referenced in this document is based upon an imaginary user who is tracking how they are devoting their daily exercise time. There are 3 physical activities that the person participates in: playing the Nintendo Wii; working out at a gym; and playing basketball. The hours spent doing each activity have been tracked over the course of a 31-day month and now the user wants to graph the data.

To view the demo dialog, launch the sample database in 4D v11 SQL, open the "Test" menu, and select "Open Demo Dialog":



There are three graphs included in the sample database. In the first graph, the total hours spent on each activity over the month are graphed. This is done by associating an object method with the drop down list, PDList1 to draw the graph onto the vGraphPict picture variable. Here is the code:

```
`Create the string for the X-axis label.
ARRAY STRING(10;times;1)
times{1}:="total"

`Arrays for holding the total time spent in each activity)
ARRAY REAL(wii_time;1)
wii_time{1}:=Sum([Phys_Time]Wii)
ARRAY REAL(gym_time;1)
```

```
gym_time{1}:=Sum([Phys_Time]Gym)
ARRAY REAL(bas_time;1)
bas_time{1}:=Sum([Phys_Time]Basketball)


GRAPH(vGraphPict;1;times;wii_time;gym_time;bas_time)
GRAPH SETTINGS(vGraphPict;0;0;0;0;False;False;True;"Wii";"Gym";"Basketball")
```

Note the arguments in the second to last line of the code:

```
GRAPH(vGraphPict;1;times;wii_time;gym_time;bas_time)
```

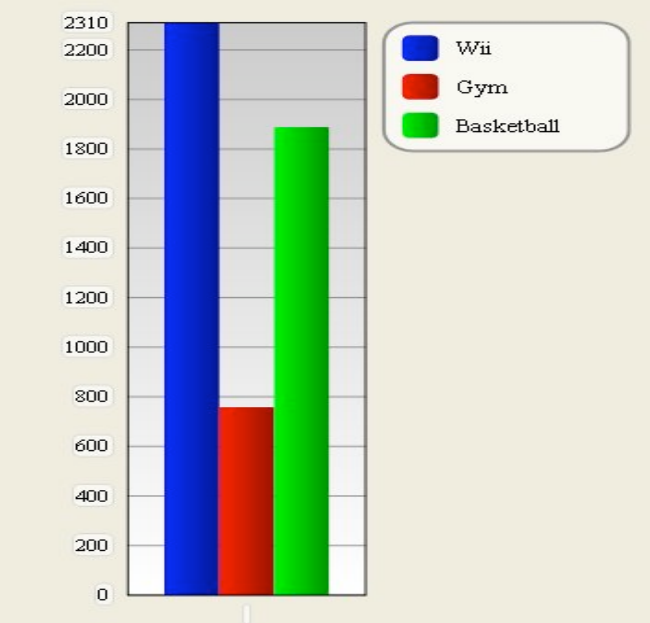
The graph style is set to 1 and thus the resultant style will be a standard column – or bar – graph. Other styles are available when using numbers 1 through 8. The last three parameters declare what datasets to graph. In this case they are the three time totals calculated earlier in the code, time spent on the Wii, at the gym and playing basketball.

Now note the settings in the last line of code:

```
GRAPH SETTINGS(vGraphPict;0;0;0;0;False;False;True;"Wii";"Gym";"Basketball")
```

The four zeros indicate the minimum X-value, maximum X-value, minimum Y-value and maximum Y-value of the graph. When set to zero, default values are used. The first "False" argument indicates that the graph will not use a proportional x-axis; instead it will use a normal x-axis. The second "False" indicates that no x-axis lines will be drawn. The "True" argument indicates that y-axis lines will be drawn. Last, the graph legend items are named. These need to be in the same order as the datasets used in the GRAPH command were called.

Here is the resulting graph:

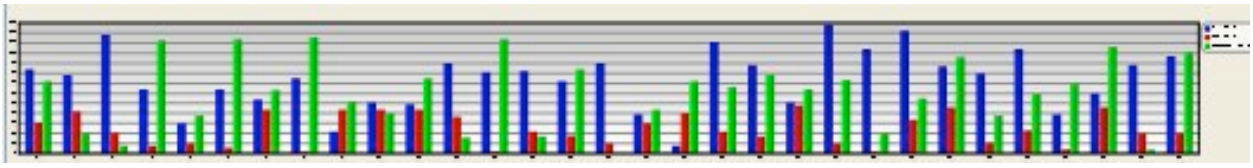


Note that the SVG graph offers a cleaner, more modern look than the older 4D Chart style. If this newer look does not suit a user's needs, the old graph look is still accessible by placing a 4D Chart area on the form.

There are two remaining graphs in the sample database: One graph charts the average time spent on each activity; and the second graph shows how much time was spent on each activity per day. Each day is printed along the x-axis and the amount of time spent on each activity is graphed per day. The algorithms that determine these values vary per graph, but the way the SVG engine is utilized is the same. Try selecting the other two graphs by changing the dropdown menu's value to see the differences.

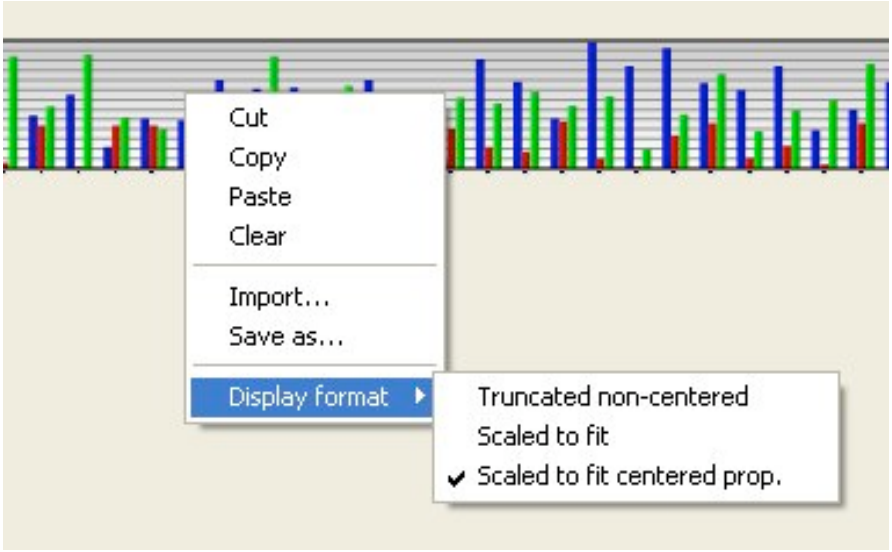
Contextual Menus and Exporting

Select "Total Days" from the drop down menu to display the time spent on each activity on each day. An SVG graph similar to the one below should show up:

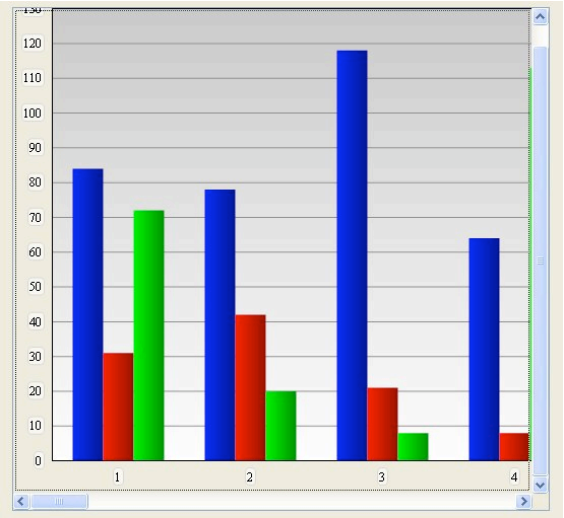


This is completely illegible! Fortunately, it is not necessary to go back into design mode to adjust how this graph is displayed.

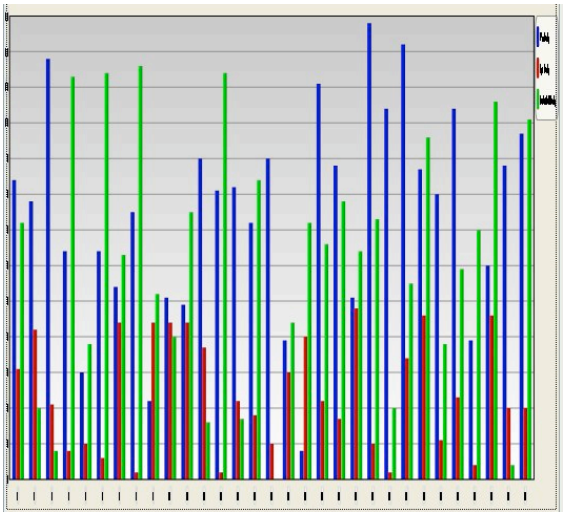
An immediate advantage to rendering images in SVG is the ability to access a contextual menu. Simply hover the mouse pointer over the graph and right-click (Windows) or Control + click (Mac OS) the image. The following contextual menu is now accessible:



There are standard “Cut”, “Copy”, “Paste” and “Clear” menu choices available to use. For the sake of remedying the display problem, expand the “Display format” choice. Try the three different formats. An example of the “Truncated non-centered” and “Scaled to fit” formats are shown below:



(a)



(b)

In image (a), the “Truncated non-centered” format allows the use of scrollbars. This allows users to view detailed information such as the graph labels along the x and y-axis, but it does not do a good job of showing an overall picture. The image in (b) does show the overall picture since the graph is scaled to fit inside the space allotted to the variable. It is also in a more manageable size than “Scaled to fit centered prop.”, but due to the image distortion, is still impossible to read.

There is one more feature that can be used to grant the user the best of both worlds. Right-click (Win) or Control + click (Mac) the graph. Select “Save as…”.

In the save dialog box that shows up, the user is given the choice to save the image as any type available on their local machine, including SVG. Why is this important? Now the user has many other image types at their disposal. For instance, they can save the image and open it using an internet browser. Zooms and scrolls are an ease using any ubiquitous browser. Furthermore, their graph is now saved for future use on the disk, in or out of 4D.

Another Way to use SVG...

When testing the application, notice that the box in the top left hand corner of the form changes to display a title for each of the different graphs. This could have been done using normal 4D text commands, but was done here using an SVG image. The code can be found in the project method, “graph_title”. The relevant snippet of code is printed here for convenience:

```
C_PICTURE(vPicHello)
$svg:=DOM Create XML Ref("svg")
```

```
$ref:=DOM Create XML element($svg;"text";"font-weight";"bold";"font-
size";18;"fill";"blue")
DOM SET XML ELEMENT VALUE($ref;$label)
DOM EXPORT TO PICTURE($svg;vPicHello;Copy XML Data Source )
DOM CLOSE XML($svg)
```

The first few lines of this code set up the XML tree and create an XML element, a "text" element. The formatting of this element (bolding, sizing and color) is all done using standard SVG code. More information on formatting the font can be found at:

http://www.w3.org/TR/SVG11/fonts.html

The last three lines set the element's value. In DOM SET XML ELEMENT VALUE, $label varies depending on the graph being displayed. This is how the value of the text is decided. The DOM EXPORT TO PICTURE command is used to save the SVG image into the picture variable, "vPicHello". The last parameter in this command lets 4D keep a copy of the XML tree with the picture for future use. The last command, DOM CLOSE XML, frees the memory that the XML object occupied.

Here is what the output looks like when displaying all days:

It is possible to modify the XML tree that was created from either of the two methods above. A presentation for manipulating XML can be found using the different tech notes available at:

http://www.4d.com/support/technotes.html

Conclusion

This technical note described the general concepts of creating SVG graphs using data from records. Two different methods for creating an SVG file were presented both in the context of concepts that are not new to the 4D design environment: Using GRAPH and a picture variable, and creating an XML element. The information above should shed a little light for the developer to utilize SVG in their databases.