

SVG – SCALABLE VECTOR GRAPHICS.....	1
DEVELOPMENT HISTORY	2
FEATURES OF SVG.....	2
SUPERIOR COLOR CONTROL	2
ZOOMING.....	2
TEXT-BASED FILES.....	2
INTERACTIVITY AND INTELLIGENCE	3
SVG DOCUMENT OBJECT MODEL (DOM).....	3
SVG PRE-CONDITIONS	3
THE APPLICATION DEMO	3
SVG AND QUICKTEST.....	4
FAST VIEW ON THE SVG OBJECT	4
Problematic issues	6
THE SVG OBJECT.....	6
INTERACTION BETWEEN SCRIPTS IN HTML AND SVG.....	7
The Source Code	8
Retrieving Data.....	13

SVG – Scalable Vector Graphics

Scalable Vector Graphics (**SVG**) is an **XML** markup language for describing two - dimensional vector graphics, both static and animated, and either declarative or scripted. It is an open standard created by the World Wide Web Consortium.

SVG allows three types of graphic objects:

- Vector graphic shapes (e.g. paths consisting of straight lines and curves, and areas bounded by them)
- Raster graphics images / digital images
- Text

Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. Text can be in any **XML** namespace suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility.

SVG drawings can be dynamic and interactive. The Document Object Model (**DOM**) for **SVG**, which includes the full **XML DOM**, allows straightforward and efficient vector graphics rich set of event handlers such as onmouseover and onclick can be assigned to any **SVG** graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on **SVG** elements and other **XML** elements from different namespaces simultaneously within the same web page.

If storage space is an issue, **SVG** images can be saved with gzip compression, in which case they may be called "SVGZ files". Because **XML** contains verbose text, it tends to compress very well and these files can be much smaller. Often however the original vector-file (**SVG**) is already smaller than the rasterised version.

Development history

SVG was developed by the W3C **SVG** Working Group starting in 1998, after Macromedia and Microsoft introduced Vector Markup Language (**VML**) whereas Adobe Systems and Sun Microsystems submitted a competing format known as PGML. The working group was chaired by Chris Lilley of the W3C.

- **SVG** 1.0 became a **W3C** Recommendation on 2001-09-04.
- **SVG** 1.1 became a **W3C** Recommendation on 2003-01-14. The **SVG** 1.1 specification is modularized in order to allow subsets to be defined as profiles. Apart from this, there is very little difference between **SVG** 1.1 and **SVG** 1.0.
- **SVG** Tiny and **SVG** Basic (the Mobile **SVG** Profiles) became **W3C** Recommendations on 2003-01-14. These are described as profiles of **SVG** 1.1.
- **SVG** Tiny 1.2 and **SVG** Full 1.2 are both currently **W3C** Working Drafts. **SVG** Tiny 1.2 was initially released as a profile, and later refactored to be a complete specification, including all needed parts of **SVG** 1.1 and **SVG** 1.2. On 2006-07-21 a new draft of the **SVG** Tiny 1.2 specification was released. A similarly refactored draft for **SVG** 1.2 Full has not yet been released.

Features of SVG

- **SVG** is a small file size.
- On average, **SVG** files are smaller than other Web graphic formats, such as JPEG and GIF, and are quick to download.

Superior color control

SVG offers a palette of 16 million colors and supports ICC color profiles, sRGB, gradients, and masking.

Zooming

Users can magnify an image up to 1,600% without sacrificing sharpness, detail, or clarity. Text stays text in **SVG**, images remains editable (within the source code) and, more importantly, **SVG** is searchable (unlike in raster and binary counterparts). There are no font or layout limitations, and users always see the image the same way you do.

Text-based files

An **SVG** file is text-based, not binary. It is a "human readable" format much like **HTML**. Even a beginner can look at **SVG** source code and immediately make sense of the descriptive content relative to the graphic representation.

Interactivity and intelligence

Since **SVG** is **XML**-based, it offers unparalleled dynamic interactivity. **SVG** images can respond to user actions with highlighting, tool tips, special effects, audio, and animation.

SVG Document Object Model (DOM)

The SVG DOM can be seen in the following address:
<http://www.w3.org/TR/SVG/svgdom.html>

SVG Pre-Conditions

To practice the examples on this chapter you first must download the Adobe SVG Viewer. In **QuickTest** you must check the **ActiveX** and **Web** addin. You can download the Adobe SVG Viewer plugin from the following address:
<http://www.adobe.com/svg/viewer/install/main.html>

The Application Demo

To practice the Visual Building Search demo please download the demo from the following address: <http://www.adobe.com/svg/demos/vbs/html/frameset.html>



Figure 1 - Visual Building Search

SVG and QuickTest

The **SVG** is an **ActiveX** element, so, when starting **QuickTest** on your computer, don't forget to check the **Web** and **ActiveX** options, in the add-in manager.



Figure 2 - QuickTest Addin Manager

Fast View on the SVG Object

To accomplish a fast view on the object please click the **QuickTest** Object Spy button on the **QuickTest** toolbar or select Tools > Object Spy; then select to point finger button and indicate one of the **SVG** panels.

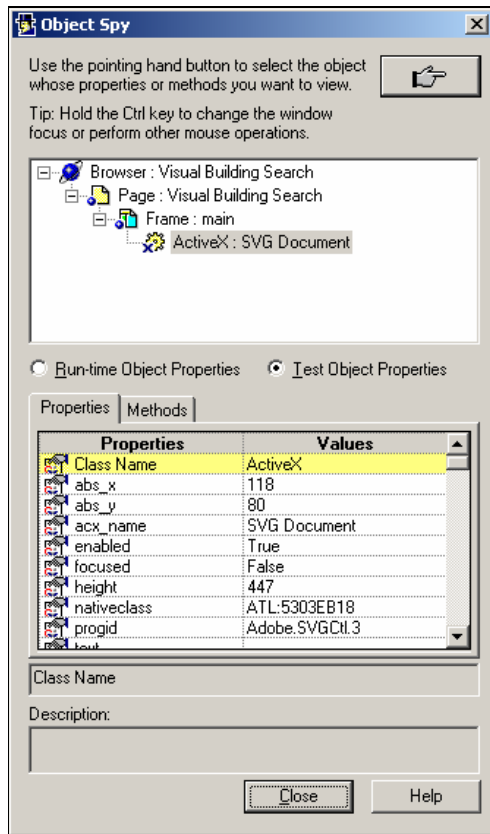


Figure 3 - Spying the SVG

As you can see the **SVG** is an **ActiveX** technology; it is identified as an **ActiveX** object of type **SVG** Document.

So, how can we work with this ActiveX?

The solution is by using the run-time methods and properties.

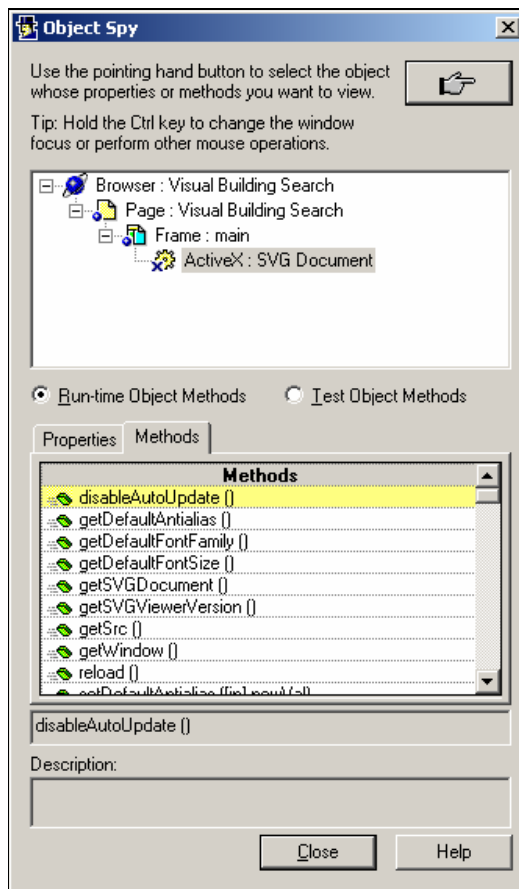


Figure 4 – SVG Run-time methods

Problematic issues

- There are a few methods to retrieve information from the **SVG** document.
- There are hundreds of properties, complex hierarchies and a lot of research work.
- This is a **Java** technology and **QuickTest** uses **VBScript**, is difficult to watch some object properties and methods at run-time.
- Run-time methods are good for application developers, as automation developers, activating the run-time method can cause damage or unexpected results to the current application session.
- As automation developers we want to be as close as possible, to human users, by simulating all user operation (mouse and keyboard).
- I found that by using run-time methods to set values and change the **SVG**, does not update the **XML** file (**SVGDocument**) attached to the application.

The SVG object

There are 2 important primary methods, to retrieve the **SVG** objects

1. `getSVGDocument()`

2. getWindow()

The **getSVGdocument** Returns the **SVGDocument** object for the referenced **SVG** document. It means that it return the attached **XML** file of the **SVG** technology. We don't want to test the **SVG** technology; we want to test our application that uses the **SVG** technology.

So the method should not be used by automation developers.

Please add the **SVG ActiveX** to your object repository, note that are to frames.

The left frame is the navigation frame, and the right one is the main frame.

You can change the logical names of the objects, to be more familiar with the objects types.

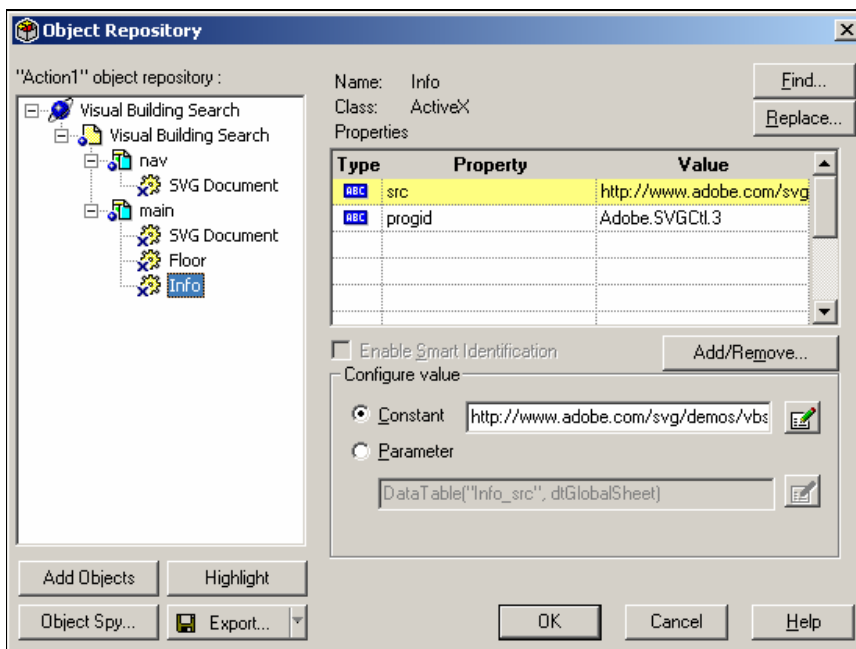


Figure 5 - The Object Repository

Interaction between Scripts in HTML and SVG

Each **SVG** embedded in **HTML** has its own **ECMAScript** "window" (global) object which can be accessed using **getWindow()** call. All functions and variables defined on global level in **SVG** can be accessed through this object. This is very similar to accessing objects in different **HTML** frames. Vice versa, **HTML** "window" (global) object can be accessed through parent variable from **SVG** scripts.

```
Option Explicit
Dim oSvgWin, oSVG
'--- Return the ActiveX object
Set oSVG = Browser("VBSearch").Page("VBSearch").Frame("main").ActiveX("SVGDoc").Object
'--- Return the SVG window
Set oSvgWin = oSVG.getWindow()
Stop
```

Run the above code, and add to the Watch expressions in **QuickTest** debugger the variable **oSvgWin**

Name	Value
oSvgWindow	<Object>
window	<Object>
_window_impl	<Object>
toString	<Object>
alert	<Object>
confirm	<Object>
setTimeout	<Object>
clearTimeout	<Object>
setInterval	<Object>
clearInterval	<Object>
getDocument	<Object>
postURL	<Object>
getURL	<Object>
parseXML	<Object>
printNode	<Object>
setDefaultAntialias	<Object>
getDefaultAntialias	<Object>
setDefaultFontFamily	<Object>
getDefaultFontFamily	<Object>
setDefaultFontSize	<Object>
getDefaultFontSize	<Object>
getSVGViewerVersion	<Object>
disableAutoUpdate	<Object>
setSrc	<Object>
getSrc	<Object>
reload	<Object>
focus	<Object>
blur	<Object>
navigator	<Object>
thisInit	<Object>
onF1r	<Object>
offF1r	<Object>
bldgNav	<Object>
evt	Empty

Figure 6 - getWindow() reference object

The Source Code

This is the **SVG** document under the navigation frame. We need to retrieve the positions of the *buttons West Tower, East Tower* and the *floors*, for this, we need to know the objects id's. We also need the *Display* object, to retrieve the current data. As I said before, every **SVG** Document has an **XML** file. To retrieve the buttons id's we need to see the **XML** source for this **SVG** document. To retrieve the **XML** file, we need the access to the source code of the **SVG** document.

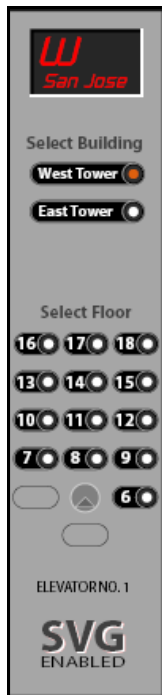


Figure 7 – The Panel Art object

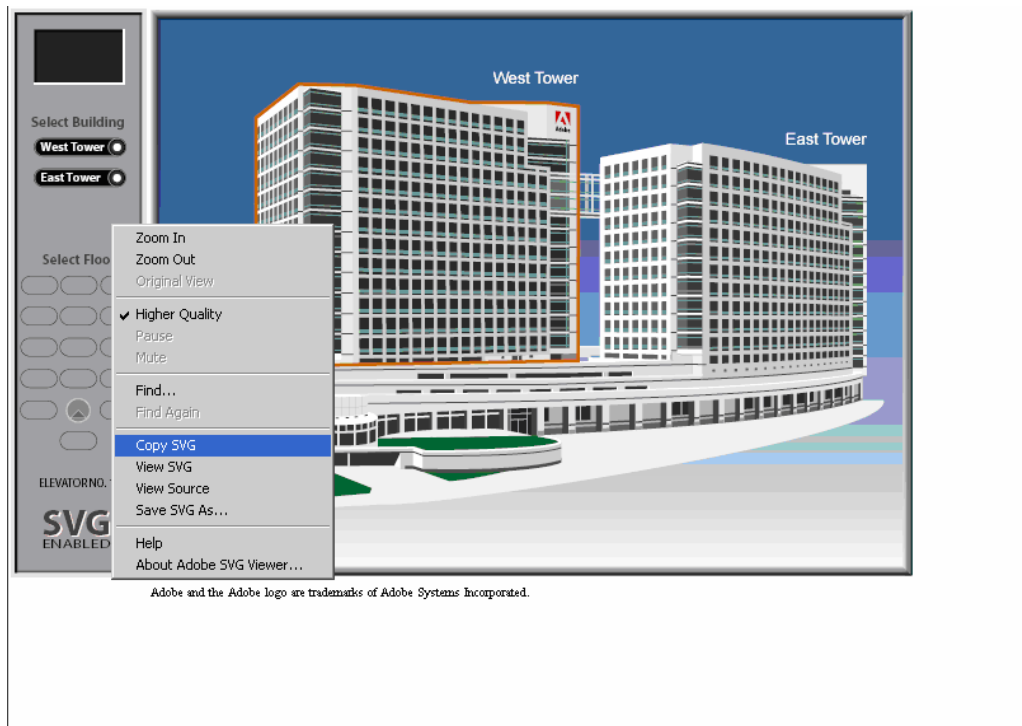


Figure 8 - Retrieving the source code

Perform a right-click on the Navigator **SVG** panel and select Copy **SVG** as shown above in figure

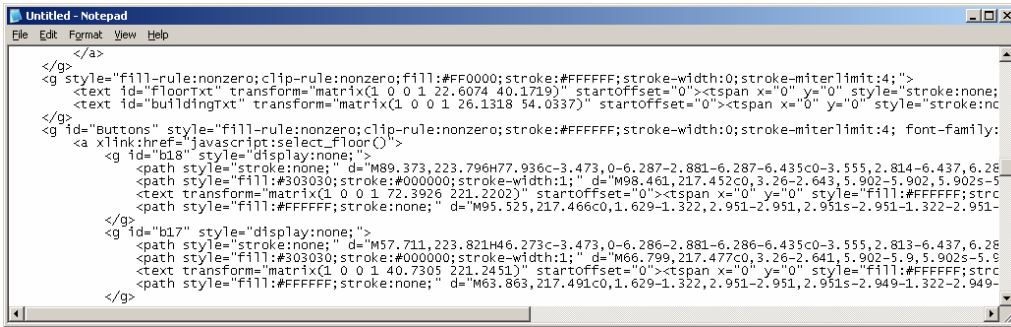


Figure 9 - Xml view on notepad

Or If you have Altova XMLSpy <http://www.altova.com/>

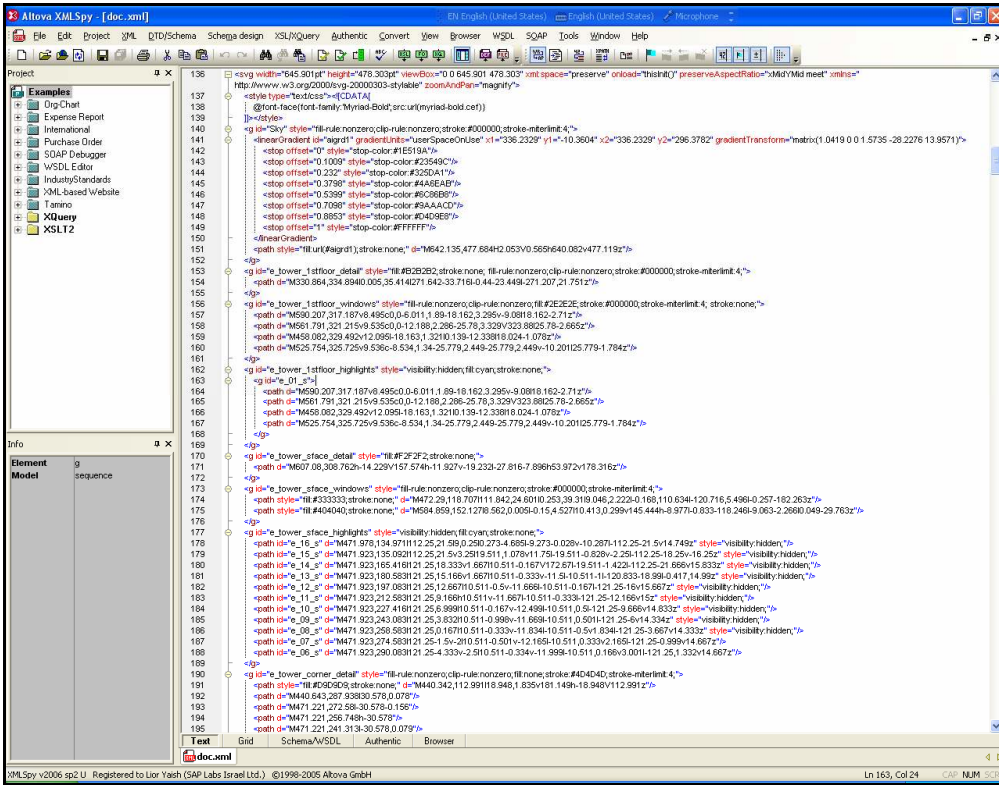


Figure 10 - Altova XMLSpy, Text view

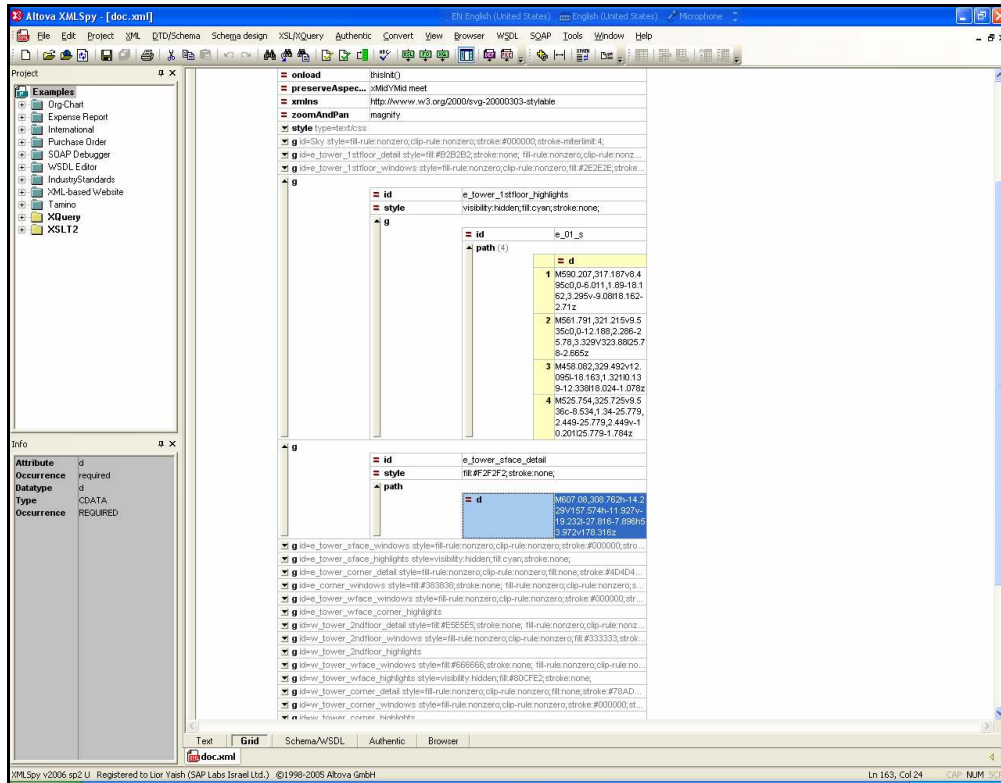
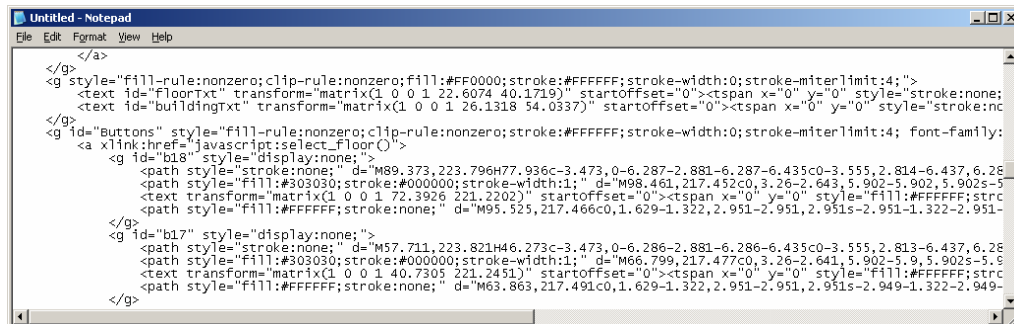


Figure 11 - Altova Grid View



Note the follow id's floorTxt, buildingTxt, under buttons tag → b18, b17 etc. Those represent the button objects; so how do I retrieve the button **b17** object?

```
With Browser ("VBSearch") .Page ("VBSearch") .Frame ("nav")
    Set b17 =.ActiveX ("SVGDoc") .Object.getSVGDocument ().getElementById ("b17")
End With
```

Since that we already know the id name of the tag, use the method *getElementById* from the **SVG** document as shown above. Run the above code, add a break point and add the **b17** object to the **Quicktest** Watch Expression panel

Name	Value
[-] b17	<Object>
nodeName	"g"
parentNode	<Object>
nodeType	1
ownerDocument	<Object>
firstChild	<Object>
addEventListener	<no value>
appendChild	<no value>
viewportElement	<Object>
namespaceURI	"http://www.w3.org/2000/svg-20000303-stylable"
hasAttributes	True
dispatchEvent	<no value>
normalize	Empty
getElementsByTagName	<no value>
getCTM	<Object>
tagName	"g"
prefix	Null
id	"b17"
getElementsByTagNameNS	<no value>
lastChild	<Object>
previousSibling	<Object>
removeAttributeNS	<no value>
setAttributeNS	<no value>
getBBox	<Object>
nodeValue	Null
nextSibling	<Object>
removeEventListener	<no value>
cloneNode	<no value>
hasAttributeNS	<no value>
hasChildNodes	True
attributes	<Object>
setAttributeNode	<no value>
removeAttribute	<no value>
getAttributeNS	<no value>
getAttributeNode	<no value>
hasAttribute	<no value>
localName	"g"
style	<Object>
replaceChild	<no value>
removeChild	<no value>
isSupported	<no value>
getAttribute	<no value>
setAttribute	<no value>
removeAttributeNode	<no value>
setAttributeNodeNS	<no value>
getAttributeNodeNS	<no value>
insertBefore	<no value>
childNodes	<Object>

Figure 12 - b17 object properties and methods

As you see, there's a lot of run-time information, also a lot of unnecessary information for the automation task. To retrieve additional information about the objects, you need to know the basics of **XML**. For the automation task, we need only to know the current position of the button. The method *getBBox* will fit for the mission.

Name	Value
[-] b17.getBBox	<Object>
x	39.987
y	210.949
width	26.812
height	12.87201

Figure 13 - getBBox method

```
nX = b17.getBBox().x
```

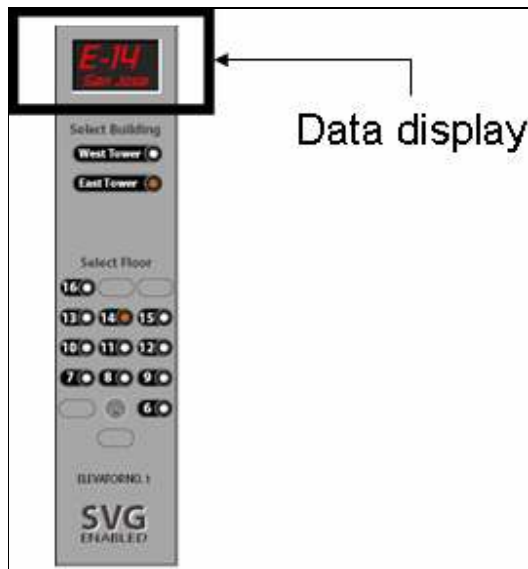
```

nY = b17.getBoundingBox().y
nW = b17.getBoundingBox().width
nH = b17.getBoundingBox().height
'--- Calculating the middle coordinates to perform click
nPosX = Round(nX + (nW / 2), 0)
nPosY = Round(nY + (nH / 2), 0)
'--- Using the Click method
With Browser ("VBSearch").Page ("VBSearch").Frame ("nav")
    .ActiveX ("SVGDoc").Click nPosX, nPosY, micLeftBtn
End With

```

Note: Remember that **SVG** is implemented in **Java**. **Java** is a case sensitive programming language; the same method will work for all other buttons.

Retrieving Data



```

<g style="fill-rule:nonzero;clip-rule:nonzero;fill:#FF0000;stroke:#FFFFFF;stroke-width:0;stroke-miterlimit:4;">
  <text id="floorTxt" transform="matrix(1 0 0 1 22.6074 40.1719)" startOffset="0"><tspan x="0" y="0"
    style="stroke:none; font-family:'RussellSquare-Oblique'; font-size:26;">E-14</tspan></text>
  <text id="buildingTxt" transform="matrix(1 0 0 1 26.1318 54.0337)" startOffset="0"><tspan x="0" y="0"
    style="stroke:none; font-family:'RussellSquare-Oblique'; font-size:12;">San Jose</tspan></text>
</g>

```

We want to retrieve at run-time of course the values *E-14* and *San Jose*. Between the **g** tags there's 2 additional children tags; text *id=floorTxt* and text *id=buildingTxt*

```

With Browser ("VBSearch").Page ("VBSearch").Frame ("nav").ActiveX ("SVGDoc")
    Set oTxt = .Object.getSVGDocument().getElementById ("buildingTxt")
End With

```

The next child tag (not attribute) is the *tspan*

Name	Value
oBuildTxt.getFirstChild()	<Object>
getNumberOfChars	8
ownerDocument	<Object>
replaceChild	<no value>
setAttributeNS	<no value>
localName	"tspan"
parentNode	<Object>
getEndPositionOfChar	<no value>
getRotationOfChar	<no value>
nodeName	"tspan"
getStartPositionOfChar	<no value>
hasChildNodes	True
getExtentOfChar	<no value>
getAttributeNodeNS	<no value>
attributes	<Object>
nodeValue	Null
normalize	Empty
removeAttributeNS	<no value>
firstChild	<Object>
setAttributeNode	<no value>
previousSibling	<Object>
tagName	"tspan"
id	""
isSupported	<no value>

Figure 14 – The *span* Tag

In the **XML** tag we see that, the text **San Jose** is not an attribute of *tspan* But, is another child of *tspan*, the number of *tspan* children is 1 as shown in **Figure 15**

Name	Value
id	""
isSupported	<no value>
cloneNode	<no value>
hasAttributeNS	<no value>
childNodes	<Object>
item	<no value>
length	1
addEventListener	<no value>
getComputedTextLength	52.66718
getSubStringLength	<no value>
style	<Object>
dispatchEvent	<no value>
viewportElement	<Object>
removeAttribute	<no value>
getElementsByTagNameNS	<no value>
appendChild	<no value>
prefix	Null
removeEventListener	<no value>
removeChild	<no value>
getElementsByTagName	<no value>
removeAttributeNode	<no value>
hasAttribute	<no value>
lastChild	<Object>
getCharNumAtPosition	<no value>

Figure 15 – *tspan* children

Under *getFirstChild* method of *tspan* tag we see that the data attribute is the string that we just need, as shown in below

Name	Value
[-] firstChild	<Object>
previousSibling	<Object>
attributes	<Object>
data	"San Jose"
removeChild	<no value>
nodeValue	"San Jose"
deleteData	<no value>
splitText	<no value>
insertData	<no value>
replaceData	<no value>
nodeName	"#text"
nodeType	3
parentNode	<Object>
appendChild	<no value>
prefix	Null
insertBefore	<no value>
namespaceURI	""
lastChild	<Object>
hasAttributes	False
isSupported	<no value>
nextSibling	<Object>
childNodes	<Object>
firstChild	<Object>
normalize	Empty
substringData	<no value>
appendData	<no value>
length	8
ownerDocument	<Object>
localName	Null
cloneNode	<no value>
hasChildNodes	False
replaceChild	<no value>
setAttributeNode	<no value>
previousSibling	<Object>
tagName	"tspan"
id	""
isSupported	<no value>
cloneNode	<no value>
hasAttributeNS	<no value>
[-] childNodes	<Object>
addEventListener	<no value>
getComputedTextLength	52.66718
getSubStringLength	<no value>
[-] style	<Object>
dispatchEvent	<no value>
[-] viewportElement	<Object>

Figure 16 – firstChild Object

```
sData = oTxt.getFirstChild().getFirstChild().data
or
sData = oTxt.getFirstChild().getFirstChild().nodeValue
or
oTxt.getFirstChild().childNodes.item(0).nodeValue
```

Example1

In the following example code we click the **West Tower** button The expected result is **W San Jose** then click the **East Tower** button and then the **floor 13** button, the expected result now is **E-13 San Jose** To calculate the center coordinates I wrote a private local Sub, it is possible to put that function is an external **vbs** resource file.

The Source Code

```

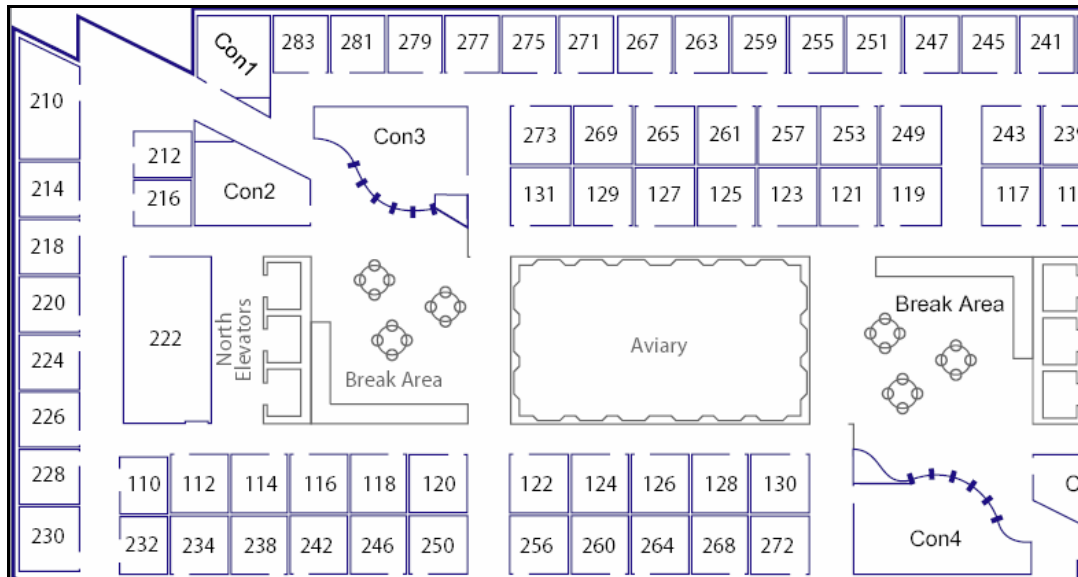
Option Explicit
Dim nPosX, nPosY
Dim oBtn, oBuildTxt, oFloorTxt
Dim sData
Private Sub GetMiddle(ByVal oItem, ByRef nClickX, ByRef nClickY)
    Dim nW, nH, nX, nY
    nX = oItem.getBoundingBox.x
    nY = oItem.getBoundingBox.y
    nW = oItem.getBoundingBox.width
    nH = oItem.getBoundingBox.height
    nClickX = Round(nX + (nW/2), 0)
    nClickY = Round(nY + (nH/2), 0)
End Sub

With Browser("VBSearch").Page("VBSearch").Frame("nav").ActiveX("SVGDoc")
    Set oBtn = .Object.getSVGDocument().getElementById("WestTower")
    Set oBuildTxt = .Object.getSVGDocument().getElementById("buildingTxt")
    Set oFloorTxt = .Object.getSVGDocument().getElementById("floorTxt")
    GetMiddle oBtn, nPosX, nPosY
    '--- Clicking the West tower button - expected result "W San Jose"
    .Click nPosX, nPosY, micLeftBtn : Wait 3
    sData = oBuildTxt.getFirstChild().getFirstChild().data
    If StrComp(sData, "San Jose") = 0 Then
        Reporter.ReportEvent micPass, "Bld. Text", sData
    Else
        Reporter.ReportEvent micFail, "Bld. Text", "Exp: San Jose; Act: " & sData
    End If
    sData = oFloorTxt.getFirstChild().getFirstChild().data
    If StrComp(sData, "W") = 0 Then
        Reporter.ReportEvent micPass, "Floor Text", "W"
    Else
        Reporter.ReportEvent micFail, "Floor Text", "Exp: 'W'; Act: " & sData
    End If
    '--- Clicking East Tower Floor 13 - Expected result "E-13 San Jose"
    '--- Clicking building east
    Set oBtn = Object.getSVGDocument().getElementById("EastTower")
    GetMiddle oBtn, nPosX, nPosY
    .Click nPosX, nPosY, micLeftBtn : Wait 1
    '--- Clicking button foloor 13
    Set oBtn = Object.getSVGDocument().getElementById("b11")
    GetMiddle oBtn, nPosX, nPosY, 0
    .Click nPosX, nPosY, micLeftBtn : Wait 3
    sData = oFloorTxt.getFirstChild().getFirstChild().data
    If StrComp(sData, "E-11") = 0 Then
        Reporter.ReportEvent micPass, "Floor Text", "E-11"
    Else
        Reporter.ReportEvent micFail, "Floor Text", "Exp: 'E-11'; Act: " & sData
    End If
    Set oFloorTxt = Nothing : Set oBuildTxt = Nothing : Set oBtn = Nothing
End With

```

Example2

In the following example code we click the **East Tower button** and then the **floor 13 button**, the expected result now is **E-13 San Jose**. To calculate the center coordinates I wrote a private local **Sub**, the *bOffset* flag = True returns the x and y coordinates only, it is possible to put that function in an external vbs resource file. The Program collects information about all the employees in the current floor between room number 200 and 260, the result report will be exported to an external MS-Excel file.



The Source Code

```

Option Explicit
Dim PosX, PosY, iLoop, nLoop, nRow, nFromRoomNo, nToRoomNo
Dim oBtn, oRoom, oInfo
Dim sData
Private Sub GetMiddle (ByVal oItem, ByRef nClickX, ByRef nClickY, ByVal bOffset)
    Dim nW, nH, nX, nY
    nX = oItem.getBoundingBox
    nY = oItem.getBoundingBox
    nW = oItem.getBoundingBox.width
    nH = oItem.getBoundingBox.height
    If bOffset Then
        nClickX = Round(nX + (nW/2), 0)
        nClickY = Round(nY + (nH/2), 0)
    Else
        nClickX = Round(nX, 0)
        nClickY = Round(nY, 0)
    End If
End Sub
With Browser("VBSearch").Page("VBSearch").Frame("nav").ActiveX("SVGDoc")
    '--- Clicking East Tower Floor 13

```

```

Set oBtn = .Object.getSVGDocument().getElementById("EastTower")
GetMiddle oBtn, nPosX, nPosY, True
'--- Clicking building east
.ActiveX("SVGDoc").Click nPosX, nPosY, micLeftBtn : Wait 3
'--- Clicking button floor 13
Set oBtn = .ActiveX("SVG Document").Object.getSVGDocument().getElementById("b11")
GetMiddle oBtn, nPosX, nPosY, True
.Click nPosX, nPosY, micLeftBtn : Wait 3
With Browser("VBSearch").Page("VBSearch").Frame("nav")
'--- Moving to the main SVG document
nRow = 1 : nFromRoomNo = 200 : nToRoomNo = 280
For iLoop = nFromRoomNo To nToRoomNo
Set oRoom = .ActiveX("Floor").Object.getSVGDocument().getElementById("Rm_" & i)
If Not oRoom is Nothing Then
GetMiddle oRoom, nPosX, nPosY, False
.ActiveX("Floor").MouseMove nPosX, nPosY : Wait 1
Set oInfo=.ActiveX("Info").Object.getSVGDocument().getElementById("ilttext")
If Not oInfo is Nothing Then
DataTable.LocalSheet.SetCurrentRow nRow
DataTable("RoomNo", dtLocalSheet) = i
DataTable("Name", dtLocalSheet) = oInfo.childNodes.item(0).data
Set oInfo = .ActiveX("Info").Object.getSVGDocument().getElementById("info1")
With oInfo.childNodes.item(1).getElementsByTagName("tspan")
sData = .item(0).firstChild.nodeValue
DataTable("Email", dtLocalSheet) = sData & "@sample.adobe.com"
sData = .item(1).firstChild.nodeValue
DataTable("Phone", dtLocalSheet) = sData
sData = .item(2).firstChild.nodeValue
DataTable("Role", dtLocalSheet) = sData
nRow = nRow + 1
Set oInfo = Nothing
End with
End If
End If
Wait 0,500
Next
Set oRoom = Nothing
End with
DataTable.ExportSheet "C:\Report.xls", dtLocalSheet

```