VectorDraw web Library

What is it?

A vector graphics library that is designed to not only open CAD drawings but also display generic vector objects on any platform that supports the HTML 5 standard, such as Windows, Android, IOS and Linux. It can be executed in every major web browser (Chrome, Firefox, Safari, Opera, Dolphin, Boat and more) that supports the use of canvas and Javascript, without any installation. This means that you can show your work from many formats like DXF, DWG, DGN, VDML and more (see below for VDS format), on almost every Computer, Tablet, Smartphone or Laptop out there. VectorDraw web library is written exclusively in Javascript and runs on the client side, also it contains an object model similar to that of DXF and .Net VectorDraw Framework components.

What can it do?

VectorDraw web compoment is a revolutionary new project, its features and capabilities are being increased in a rapid frequency. This means that new functions and possibilities are added all the time. To this point using the VectorDraw web libary you can demonstrate drawings in 2D and 3D, execute view operations like Panning and Zooming, get specific entities from the drawing, edit existing and insert new entities to it. Every entity supported by the .Net VectorDraw components can be displayed without problems, this means that all supported drawings from our .Net components can be exported to the new web component. Also the object format of the new library is very similar to those of DXF and VectorDraw .Net.

How can I use it?

In order to open a drawing there are only three things needed.

First the library's engine script. This script contains all the needed code in order for the drawing to be displayed in the screen. This file is the heart of the library and should not be meddled with. To ensure that it will work properly and that it will offer the level of performance it is designed to, this script file is offered obfuscated. All the appropriate usability it can offer is described in the help file, as well as in the sample projects we have created. Also keep in mind that it's impossible to offer support to a library that is not entirely created by us.

Second an HTML canvas object where all the visible parts of the drawing will be displayed.

Third a specially exported and formatted drawing that contains all the document's data as well as its different entities and their properties. To export the said document, you simply need to call the vdDocument.ExportScript() or use the ExportScript command of vdfCAD. The exported file will have a .vds suffix.

Additionally you'll find the commands.js file. In this script, many commands are implemented by us in order to help you use the library. If you open this file, you'll notice that all the methods and variables are not obfuscated at all, their names are intact. That way we want to encourage you to modify as you see fit the included code in order to best fit it to your needs.

Developers with prior knowledge of Javascript will have no problem catching up to the component's logic. However, developers with small experience in web programming will also find it extremely easy to work with this library, since it requires no more that a few basic HTML and Javascript commands in order to effectively operate.

How to use VectorDraw web Framework in an HTML page?

Since the library uses the canvas HTML object in order to display its functionality, it is safe to assume that most scenarios of the library's usage will be in an HTML page. To implement such a solution is very easy. First create a new HTML page with the default tags.

- Add a canvas object and a button. We'll assign the opening of the document on the Click event of the button. You don't have to set width and height to the canvas since these will be set later.
- Then we need to tell the page where it will find the VD web library. That can be achieved with the following command.

```
<script type="text/javascript" src='<name of script>.js'></script>
```

In order for this to work you have to make sure that the script is stored in the same directory as the .html file, using the same filename that you are using in the command above.

- Next we'll assign the opening operation to the button's onClick event. This can be done very easily by editing the button tag inside the HTML page like this

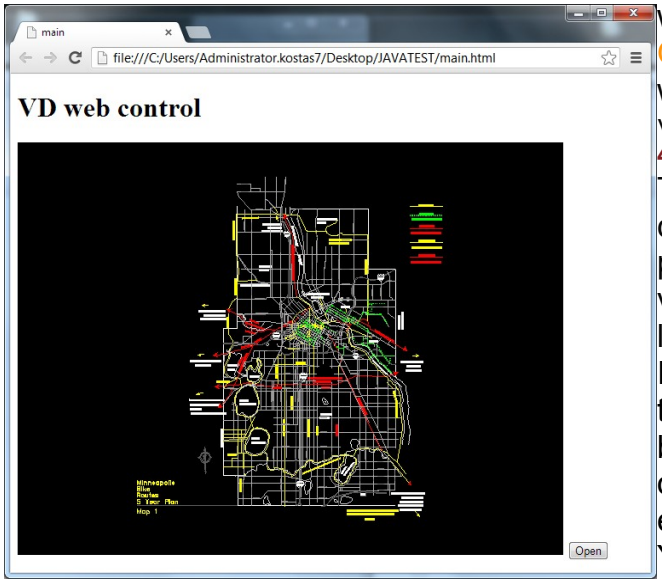
```
<button onclick="vdmanager.vdrawObject('myCanvas').SelectDocument('<document filename>')">
Open
</button>
```

The document filename is the name of the file exported from the .Net VD component. Make sure to include all the files that were exported in the same directory as the .html file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>main</title>
  <script type="text/javascript" src='vdWebControl.js'></script>
  <script type="text/Javascript">
        function vdrawInitPageLoad() {
            vdmanager.AttachCanvas('myCanvas', 640, 480);
            var vdcanvas = vdmanager.vdrawObject('myCanvas');
        }
  </script>
</head>

<body onload="vdrawInitPageLoad()">
  <div>
    <header>
      <h1>VD web control</h1>
    </header>
    <div>


        <canvas id="myCanvas"></canvas>

                <button
onclick="vdmanager.vdrawObject('myCanvas').SelectDocument('vddocument.vds')
">
        Open
        </button>

    </div>
  </div>
</body>
</html>
```

The final step is to bind the canvas object with our library. What we are doing is instructing the library to display its data on this specific canvas object. This can be done whenever we want in our page, but in this example we'll create a new Javascript function that will run whenever we open the page (on body's OnLoad event). Inside the said function we'll write this.

```
vdmanager.AttachCanvas('myCanvas', 640, 480);
```



The AttachCanvas function will resize the canvas object to whatever you have set as parameters here. Keep also in mind that the vdmanager object is defined inside the library's script, so you can call it right away.

If you did all of the above steps and run the .html file with a web browser you'll see a black square and a button on the side. Click on the button and the drawing you have exported will appear.

You did it! Your drawing and all its properties are now displayed using just a web browser!

You can now see your drawing, as well as Pan, Zoomin or Zoomout. You can also double click and set the zoom to the document's extents. Vectordraw web library is designed to be fully customizable. In that direction, our mouse and touch events are internally implemented, but their functionality can be easily overriden. That way you can either use our implementation of these events, or define them yourself. In order to do so, we'll need to see how to add an event listener to one of the library's events and add some code to it. This can be done quite easily.

- Add an event listener to the event you need to handle. Go to the vdrawInitPageLoad function we created earlier (this one runs whenever you load the page) and add this command.

vdcanvas.vdmousemove = _vdmousemove;

- Now add a new label element in your html page like this:

<label id="info3"></label>

This label will be used to display the cursor's position when inside the component.
_vdmousemove is the function that will handle how the library will operate once this event is invoked.

- Finally, before running the example we need to define the vdmousemove function. Go inside the <script> tag you have already created and write this code.

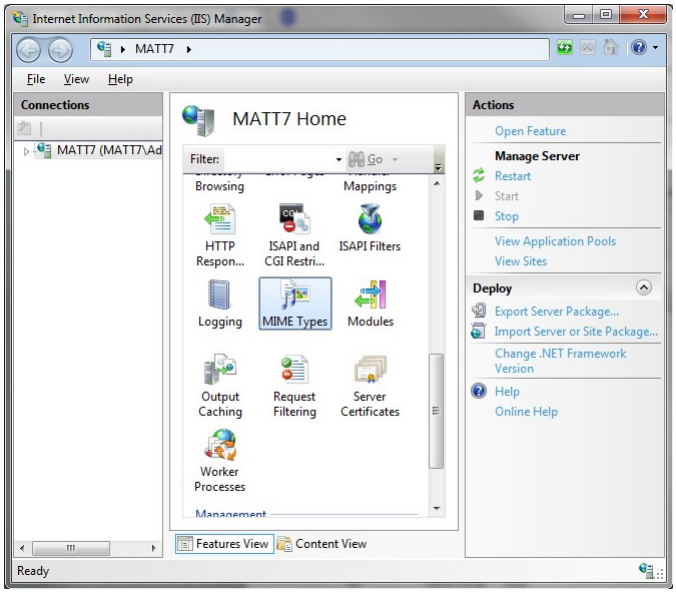
```
function _vdmousemove(e) {
     e.Cancel = true;
     if (e.mousebutton == 3) return; //right button
     if (e.mousebutton == 0) {
         e.target.canvas.style.cursor = "crosshair";
         e.target.canvas.title = "";
         var entity = GetEntityFromPoint(e.xPix, e.yPix);
         if (entity != null && entity._t != undefined) {
             e.target.canvas.style.cursor = "pointer";
             e.target.canvas.title = e.target.Fig_codeToString(entity._t) + " :
                                     " + entity.HandleId.toString();
         }
         info3.innerHTML = e.x.toString() + " , " + e.y.toString();
         return;
     }
     if (e.prevPos != null) {
         var dx = e.xPix - e.prevPos[0], dy = e.yPix - e.prevPos[1];
         e.target.scroll(dx, dy, e.target.GetDefaultTimeOutMilliseconds());
     }
 }
```

If everything was written as expected when moving the cursor around the drawing you'll see that the label displays the position of the mouse and when you stop over an entity, a tooltip shows with a small text. Additionally you can Pan the drawing. You can Pan the drawing by pressing the Left or Middle button and moving the mouse around. Let's see what exactly happens in the above function.
First you need to know what parameters the e object contains. You can find a comprehensive list of all the events and their properties at the .chm help file distributed with this “getting started” tutorial. For now we only need a few of these properties. e.mousebutton returns the mouse button that is pressed during the invoking of this event. If the property is 3 that means that the right button was pressed, so the function ends. If no button was pressed, then the property has the value of 0. In that case we change the cursor to the crosshair. We do that by editing this property e.target.canvas.style.cursor. The target property contains the vdrawobject. This object contains most of the functionality, in regard to handling of the library. For example, the open events we are handling are inside this object.

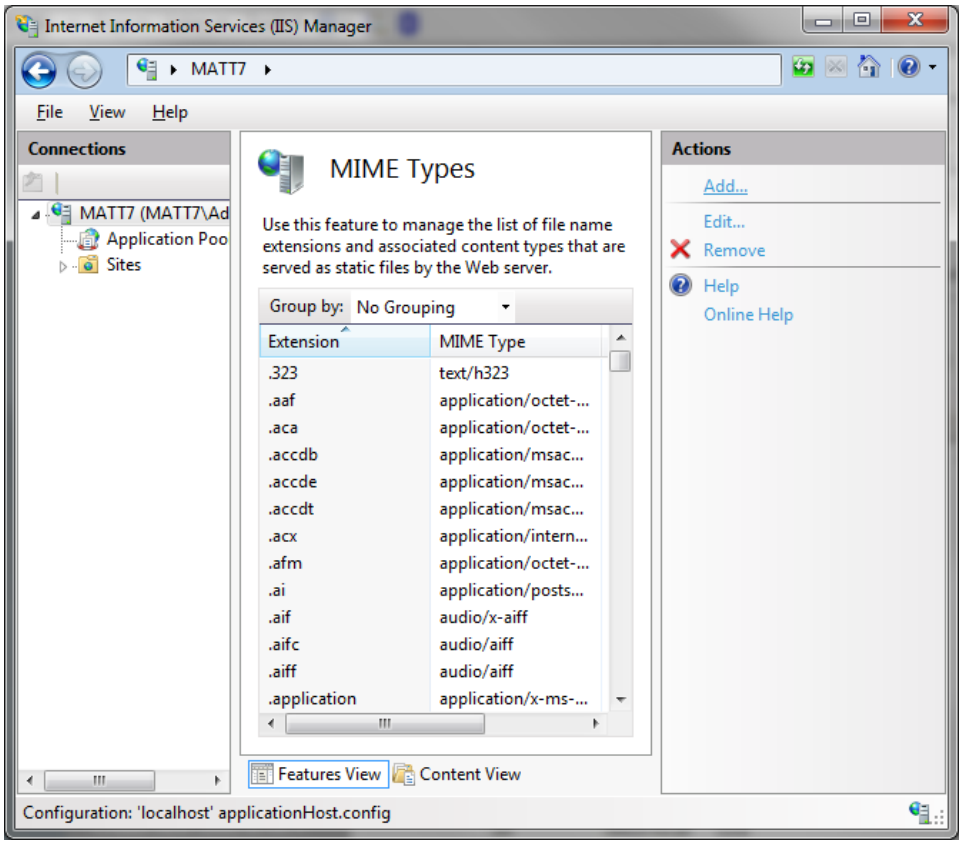
Finally we get the dx and dy of the Pan action. These two numbers show the distance our cursor has moved since the last time the event was invoked, allowing us to know how much we need to pan our drawing. To do that we call the e.target.scroll function with the x and y distance as parameters. The third parameter defines how soon after the scroll the library will redraw its contents, in milliseconds.
Similarly to the vdmousemove event you can implement every other event. Refer to the .chm help file for a full list of available events. You can also get many ideas on how to use our library by trying out many existing sample projects in our website http://www.vdraw.com/javascript-examples/vectordraw-javascript-samples/ .

How do I add a new file extension or MIME type in IIS7?
In order to be able to load .vds documents in IIS you neead a MIME type for a new file extension. MIME types are created in IIS and allow different file extensions to work in a Web browser.

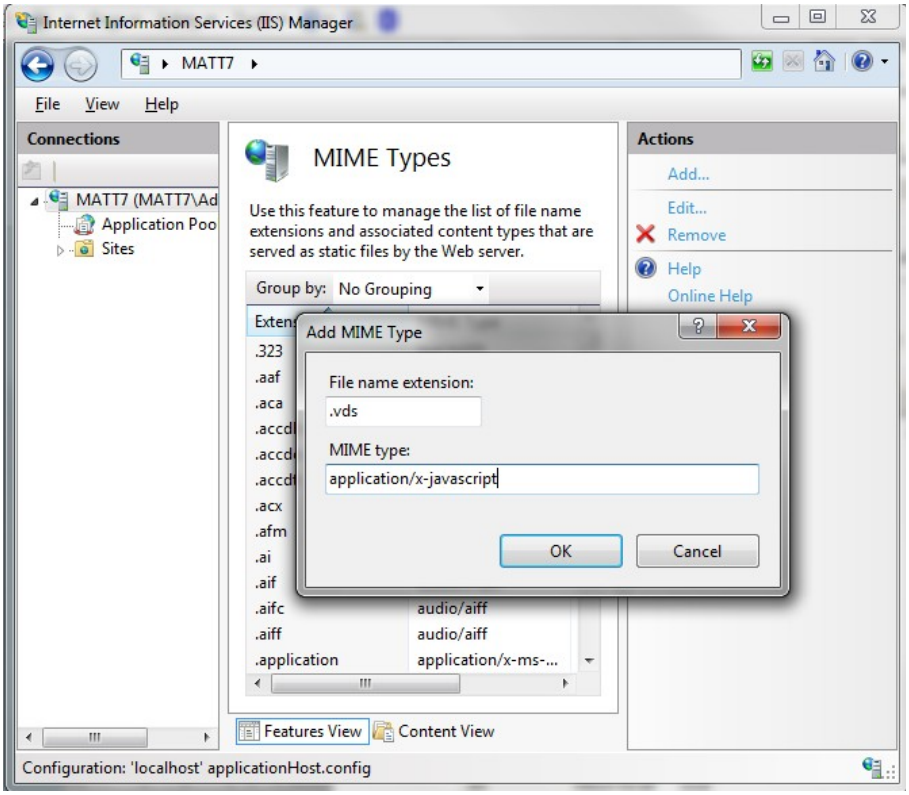
1.Log into your server through Remote Desktop Connection (optional).
2.Click Start, Programs, Administrative Tools, and select Internet Information Services (IIS) Manager
3.Click the server name. Under the IIS header, there is an icon MIME Types. Double-click the MIME Types icon to open the feature



4.On the right-hand sidebar of the feature, click Add where you can add a MIME type



5.Enter the appropriate information:



- Extension – the .vds file type extension.
- MIME type - the type of file this extension refers to (view a list of common MIME types).

6.Click OK