User Interface - Bug #4135

high dpi screens display in low dpi

07/04/2019 12:01 PM - Greg Shah

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Sergey Ivanovskiy	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to User Interface - Bug #4849: Swing font size on laptop with HiDPI i...	**New**

History

#1 - 07/04/2019 12:02 PM - Greg Shah

A customer has reported that the web client renders in a low resolution when run using a high dpi screen (e.g. a MacBook Pro).

#2 - 09/09/2019 09:44 AM - Greg Shah

- Assignee set to Sergey Ivanovskiy

#3 - 10/15/2019 05:35 AM - Sergey Ivanovskiy

It seems that canvas image smoothing (anti-aliasing) properties can cause this effect of low graphics resolution. In FWD anti-aliasing effects are switched off. Thus, turning on anti-aliasing makes text visual presentation to have smooth glyphs instead of sharp glyphs. Any way, these settings (p2j.screen.js) can control these visual effects

```
      /** Canvas 2D graphics context. */
      this.ctx = this.canvas.getContext('2d', {alpha : true});

      setImageSmoothing(this.ctx, false);

.................................................

   /**
    * Enable or disable image smoothing for the given graphics context.
    *
    * @param    {CanvasRenderingContext2D} graphicsContext
    *           The given graphics context
    * @param    {boolean} enabled
    *           The enable flag that must be true to enable image smoothing and false to disable.
    */
   function setImageSmoothing(graphicsContext, enabled)
   {
      if (graphicsContext.imageSmoothingEnabled)
      {
         graphicsContext.imageSmoothingEnabled = enabled;
      }
      else
      {
         if (graphicsContext.mozImageSmoothingEnabled)
         {
            graphicsContext.mozImageSmoothingEnabled = enabled;
         }

         if (graphicsContext.webkitImageSmoothingEnabled)
         {
            graphicsContext.webkitImageSmoothingEnabled = enabled;
         }
```

```
            if (graphicsContext.msImageSmoothingEnabled)
            {
                graphicsContext.msImageSmoothingEnabled = enabled;
            }
        }
    }
```

and if image smoothing is on, then ctx.imageSmoothingQuality = "low" || "medium" || "high"; can have an effect.
It is an issue for me how to check these functionality on high resolution screens at this moment.

#4 - 10/15/2019 05:37 AM - Sergey Ivanovskiy

- Status changed from New to WIP

#5 - 10/18/2019 11:18 AM - Sergey Ivanovskiy

Please provide the display resolution parameters for which the low graphics quality has been observed.
These parameters can be important:

```
Resolution: 1920 x 1080 pixel, diagonal_size: 21.5 inch and the calculated pixels density
PPI (pixels per inch) = SQRT(width^2 + height^2)/diagonal_size = SQRT(1920^2+1080^2)/21.5 = 102.46
```

The issue is not observed with the display parameters provided above.

From https://en.wikipedia.org/wiki/Retina_display it can be found that the retina displays have PPI more than 218. Thus they have more than twice as many pixels per inch as the display PPI above.

#6 - 10/18/2019 11:56 AM - Greg Shah

> Please provide the display resolution parameters for which the low graphics quality has been observed.

The customer saw this low resolution result on a Macbook (Pro?). I will ask for the details.

Perhaps the solution is to detect the high resolution mode and turn on anti-aliasing for this mode? As I recall, the reason we turned off anti-aliasing was because it made line drawing and text "fuzzy". Perhaps that does not occur in high resolution mode.

#7 - 10/18/2019 12:44 PM - Constantin Asofiei

Sergey, Chrome has a feature where, after you open the Developer Tools, you can test a site how it looks on different resolutions - there you can add a retina display config (2880 x 1800 at 220 pixels per inch) and see how FWD Web client behaves.

The settings to switch resolutions are bellow the address bar.

#8 - 10/18/2019 01:21 PM - Greg Shah

From the customer:

> This is seen on all high resolution displays, both Mac, Ubuntu and Windows. Although almost all modern Mac systems have high resolution (retina) displays as a standard. My MacBook Pro has a resolution of 2880x1800, but Mac OS is scaling everything 200% as if it were 1440x900 to prevent fonts, buttons etc. to be too small. The result of this is a very sharp display, on which you cannot see the pixels on a normal working distance.
>
> I also think anti aliasing will make things worse, fuzzy as you said.
>
> I have a partial screenshot attached of a screen and Virtual Box. As you can see the fonts and icons in the virtual box window are much sharper.
>
> Maybe google on 'retina java' or 'HiDPI java' can help?

The screenshot is proprietary to the customer and is not being made available for public viewing.

#9 - 10/18/2019 04:31 PM - Sergey Ivanovskiy

Constantin, thanks, I will test changes with Chrome and then provide the patch for this issue. There is another vision of this issue
https://developer.mozilla.org/en-US/docs/Web/API/Window/devicePixelRatio

#10 - 10/18/2019 05:37 PM - Sergey Ivanovskiy

It seems that for high resolution screens we should follow this example from
https://developer.mozilla.org/en-US/docs/Web/API/Window/devicePixelRatio

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

// Set display size (css pixels).
var size = 200;
canvas.style.width = size + "px";
canvas.style.height = size + "px";

// Set actual size in memory (scaled to account for extra pixel density).
var scale = window.devicePixelRatio; // Change to 1 on retina screens to see blurry canvas.
canvas.width = size * scale;
canvas.height = size * scale;

// Normalize coordinate system to use css pixels.
ctx.scale(scale, scale);
```

#11 - 10/20/2019 11:08 AM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> It seems that for high resolution screens we should follow this example from
> https://developer.mozilla.org/en-US/docs/Web/API/Window/devicePixelRatio
>
> [...]

At this moment we can't use this approach because we use these low level functions getImageData and setImageData. If the underlined offscreen canvas size is 2 times wider than its css size, then canvas scaling has no effect on the image returned by getImageData because this function works with logical pixels, but not with css pixels. We probably should think about similar approach in order to change our drawing api so that it can use 2 times wider canvas for high resolution screen and put it finally into canvas having css size 2 times less than its canvas size. All coordinates should be scaled by factor 2 and the pixel lines should be smoothed somehow. We can draw a one pixel width line but the pixel line that 4 times wider should be drawn differently than CanvasRenderer.prototype.drawSlopedLineSegment does now.

#12 - 10/20/2019 11:28 AM - Sergey Ivanovskiy

- File p2j.js.patch added

I suppose to detect high resolution screens by this media query and use it to define scaling factor 2.

```
   /**
    * Tests the screen media if it has a high resolution.
    *
    * @returns  {Boolean}
    *           True if the screen media has a high resolution.
    */
   function isHighResolution()
   {
      return window.matchMedia &&
                (window.matchMedia(
                     'only screen and (min-resolution: 192dpi), only screen and ' +
                     '(min-resolution: 2dppx), ' +
                     'only screen and (min-resolution: 75.6dpcm)').matches ||
                 window.matchMedia(
                     'only screen and (-webkit-min-device-pixel-ratio: 2), only screen and ' +
                     '(-o-min-device-pixel-ratio: 2/1), only screen and ' +
                     '(min--moz-device-pixel-ratio: 2), only screen and ' +
                     '(min-device-pixel-ratio: 2)').matches);
   }
```

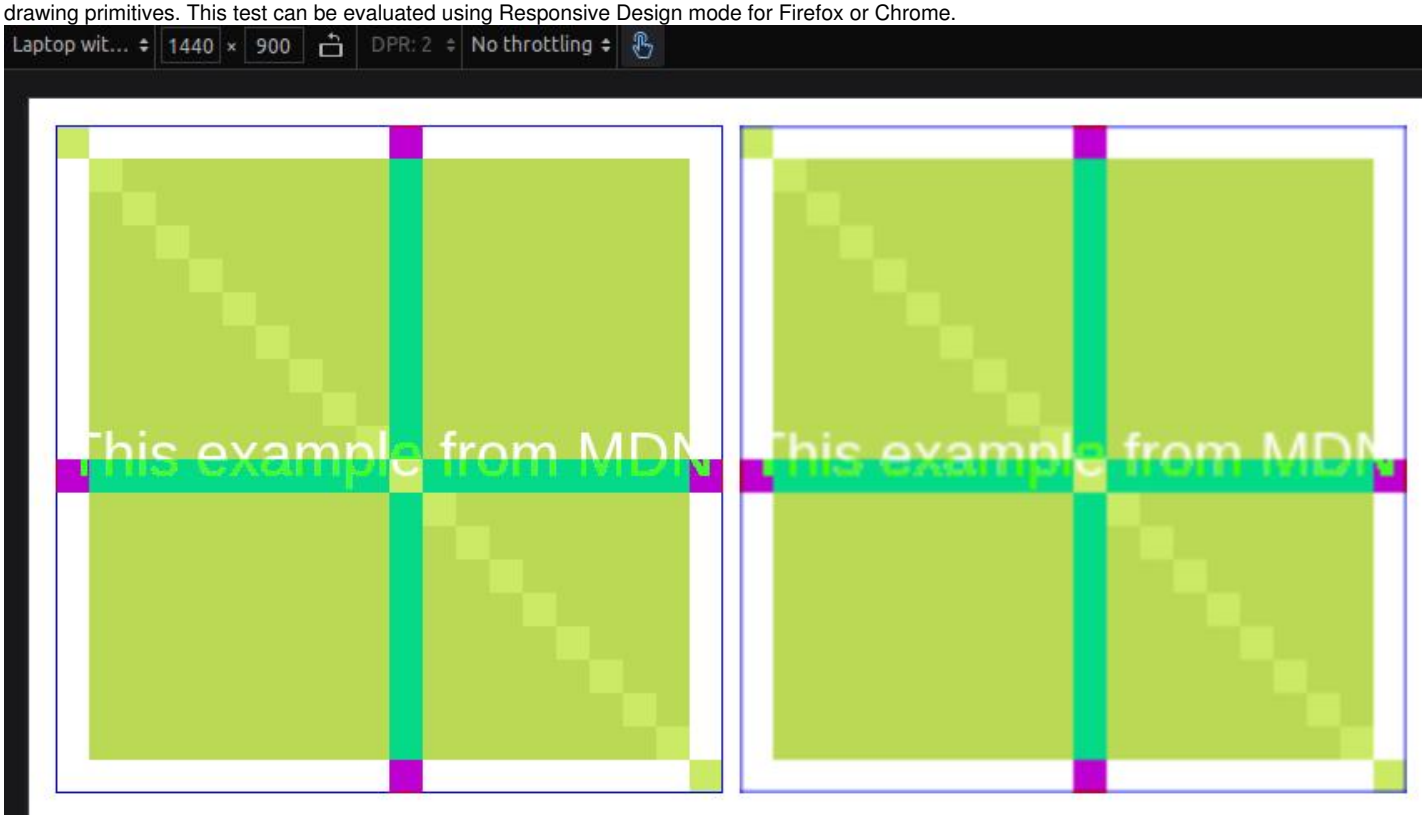
Useful references
https://github.com/strues/retinajs/
https://stackoverflow.com/questions/19689715/what-is-the-best-way-to-detect-retina-support-on-a-device-using-javascript

#13 - 10/21/2019 11:35 AM - Sergey Ivanovskiy

- File testDrawImage-1.html added

- File testDrawImage1.png added

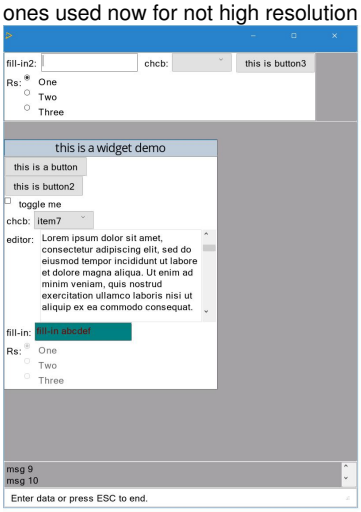
This approach should work properly but following his way I got drawing issues. Then I tested it via this standalone test page testDrawImage-1.html. It followed from this test that my issues were related to incorrectly transformed coordinates for putImageData and drawImageData and probably others drawing primitives. This test can be evaluated using Responsive Design mode for Firefox or Chrome.



#14 - 10/21/2019 06:43 PM - Sergey Ivanovskiy

- File DemoWidgetsDPIx2.png added

I fixed coordinates for drawing primitives and realized that for this task we need to support icons and images that are 2 times wider than their original ones used now for not high resolution screens. Please look at this picture. It is sharp if it is zoomed but images are small and fonts size looks small.



#15 - 10/23/2019 04:51 PM - Sergey Ivanovskiy

Created a new task branch 4135a for these changes. It needs to add new functionality to the java client that should send 2 times wider binary images than original ones if the javascript client has a high resolution screen.

#16 - 10/23/2019 05:21 PM - Greg Shah

Can you just scale the smaller images into larger ones on the Javascript side? It seems useless to have Java send more data when this is really just a scaling problem that is specific to JS.

#17 - 10/24/2019 05:23 AM - Sergey Ivanovskiy

Greg Shah wrote:

> Can you just scale the smaller images into larger ones on the Javascript side? It seems useless to have Java send more data when this is really just a scaling problem that is specific to JS.

The javascript client gets a binary image and puts it on the canvas. There is no Canvas API that can scale the binary image. I will google if there are java script open source libraries that can do this scaling.

#18 - 10/24/2019 05:46 AM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> Greg Shah wrote:
>
> > Can you just scale the smaller images into larger ones on the Javascript side? It seems useless to have Java send more data when this is really just a scaling problem that is specific to JS.
>
> The javascript client gets a binary image and puts it on the canvas. There is no Canvas API that can scale the binary image. I will google if there are java script open source libraries that can do this scaling.

drawImage() of the Canvas API can do scaling but it needs an image source but the client has only an image data that is drawn upon the existing background. The image data can be transformed into image if we put the image data into the delegated image offscreen canvas and then transforms it to image with help of this API

```
if (imageOffscreenCanvas instanceof HTMLCanvasElement)
{
   img = new Image();
   img.src = imageOffscreenCanvas.toDataURL("image/png");
}
else
{
   // OffscreenCanvas : get ImageBitmap
   img = imageOffscreenCanvas.transferToImageBitmap();
}
```

I suspect that this approach can be slower than if the client gets an original image of proper quality. The other side is that a bitmap image can lost its quality when it is scaled. If you would like to test this approach, then I will implement this way first.

#19 - 10/24/2019 09:44 AM - Greg Shah

> I suspect that this approach can be slower than if the client gets an original image of proper quality. The other side is that a bitmap image can lost its quality when it is scaled. If you would like to test this approach, then I will implement this way first.

Where would "an original image of proper quality" come from? Even if we provided high res bitmaps for all of our existing resources, the customer's application would still have many of their own low res images. Anytime a low res image is used, we will need to scale.

What was your plan as an alternative?

#20 - 10/24/2019 09:56 AM - Sergey Ivanovskiy

Now debugging this way to scale the given image on the javascript client, but as an alternative solution we can scale on the java client side within VirtualScreenImpl if the client has high resolution screen.

#21 - 10/24/2019 12:38 PM - Sergey Ivanovskiy

Committed revision 11338 (4135a) has dirty changes that show the kind of changes that should be done. If we uncommented scaling factor for getScaledImageDataFromUint8ClampedArray of p2j.screen.js module and scaling factor for drawImage of p2j.canvas_renderer.js the images became scaled properly but some artifacts could be observed. Thus the image data regions were drawn incorrect. It seems that extra data were drawn on to the canvas.
This approach still needs to draw one pixel width line correctly. Please evaluate these changes. I need to take a short time-out to think more about this issue.

#22 - 10/25/2019 05:05 AM - Sergey Ivanovskiy

In this article another method to scale images is described. https://phoboslab.org/log/2012/09/drawing-pixels-is-hard

#23 - 10/27/2019 07:28 AM - Sergey Ivanovskiy

I checked a similar approach in which the offscreen canvas remains unscaled and when it is drawn onto the screen canvas it is zoomed while the screen canvas is adapted to the same css dimension as the dimension of the offscreen canvas. We can't follow this way because all drawings and text are antialiased and don't have sharp outlines. Thus there is only one way to draw on the scaled offscreen canvas and then to copy it on to the screen canvas that has the same logical dimension as the offscreen canvas, but its css dimension equals to the dimension of the unscaled offscreen canvas. Following this approach we will get sharp outlines but it can be observed white drawing artifacts that corresponds to some clipping regions and this regions have incorrect background colors. At this moment I don't know how to resolve this issue. It can be that it is related to half pixel conception that a logical point having fractional coordinates.

I came to the idea that x and y coordinates values of a pixel should be whole numbers. If these values are not integers, then some approximations are probably applied and we can't rely on our calculations. If I understand correctly we developed sub pixel conception for these half integers, but I don't know any articles or documentations that support this conception.

Committed revision 11342 (4135a) fixed the task bar resize. Please look at demo_widgets.p with this revision in responsive design mode for Laptop with HDPI screen (DPI>=2)

#24 - 10/28/2019 09:06 AM - Greg Shah

> If I understand correctly we developed sub pixel conception for these half integers

No. The reason for the shift of a half pixel in each direction is to eliminate problems with anti-aliasing. Please see #1811-744 for the details. You can try the implementation without the shift and check the results. If I recall we implement the transform in just a single place, early in the creation/init of the canvas, so it should be easy to check.

> Following this approach we will get sharp outlines but it can be observed white drawing artifacts that corresponds to some clipping regions and this regions have incorrect background colors.

I suspect that you must have the same .5 shift transform implemented for the offscreen canvas so that the results will match.

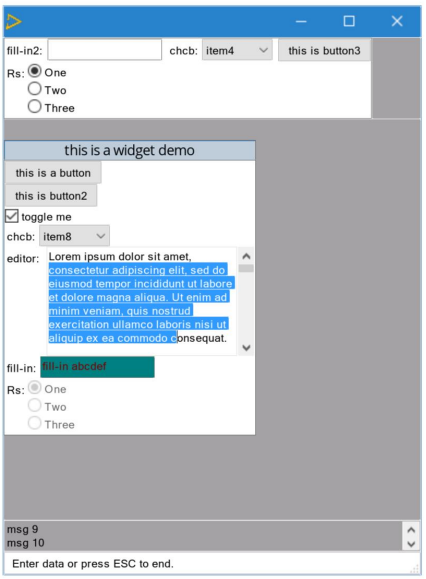
> Please look at demo_widgets.p with this revision in responsive design mode for Laptop with HDPI screen (DPI>=2)

Can you please post the screenshot here?

#25 - 10/28/2019 09:29 AM - Sergey Ivanovskiy

- File Screen Shot 2019-10-28 at 16.23.45.png added

OK. Please look at this screen shot.



#26 - 10/28/2019 09:30 AM - Sergey Ivanovskiy

- File DemoWidgets1.png added

#27 - 10/28/2019 09:31 AM - Sergey Ivanovskiy

- File deleted (Screen Shot 2019-10-28 at 16.23.45.png)

#28 - 10/29/2019 05:21 PM - Sergey Ivanovskiy

- Status changed from WIP to Review

- % Done changed from 0 to 100

#29 - 10/29/2019 05:29 PM - Sergey Ivanovskiy

Committed revision 11343 (4135a) tried to fix the drawing issue from #4135-25. The root cause seems to be related to the fact that two neighbour lines become separated by a new line on the scaled canvas so clipping regions can have gaps that produce such drawing artifacts. I tried to fix this issue by extending line width but it seems that the most correct solution is to draw the same drawing by scaling its dimension.

#30 - 11/06/2019 04:49 AM - Sergey Ivanovskiy

The correct way to draw on the high resolution screen is to draw scaled drawings on the java side.
The java script client can't fix all possible artifacts that can be observed after scaling due to scaling produces gaps that should be filled but it can be done only on the java side for each widget class separately. If we have a bitmap image of low dimensions, then we can't zoom it preserving the image quality unless the bitmap can be represented as a sequence of vector drawings. Unfortunately, we have bitmap drawings. Zooming produces gaps that must be filled. Two parallel lines of one pixel width become separated by a line of undefined color.

#31 - 11/06/2019 06:19 AM - Constantin Asofiei

Can you post an image with e.g. Hotel GUI on a normal screen and on a HDPI screen?

Related to your current scaling approach: the issue in #4135-30, is it related to drawing 4GL images, or actual FWD primitives (line strokes, text, etc)?

#32 - 11/06/2019 06:33 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> Related to your current scaling approach: the issue in #4135-30, is it related to drawing 4GL images, or actual FWD primitives (line strokes, text, etc)?

Yes, it is related to actual FWD primitives because they are drawn on the java side by using knowledge about screen resolution. Some widget drawings take into account pixel widths or heights. The matter of this issue can be observed for the simple example of two lines but you can use another example of two neighbour areas to observe that this way doesn't work or to be more precise can't work properly.

> Can you post an image with e.g. Hotel GUI on a normal screen and on a HDPI screen?

Yes, I will post these images later.

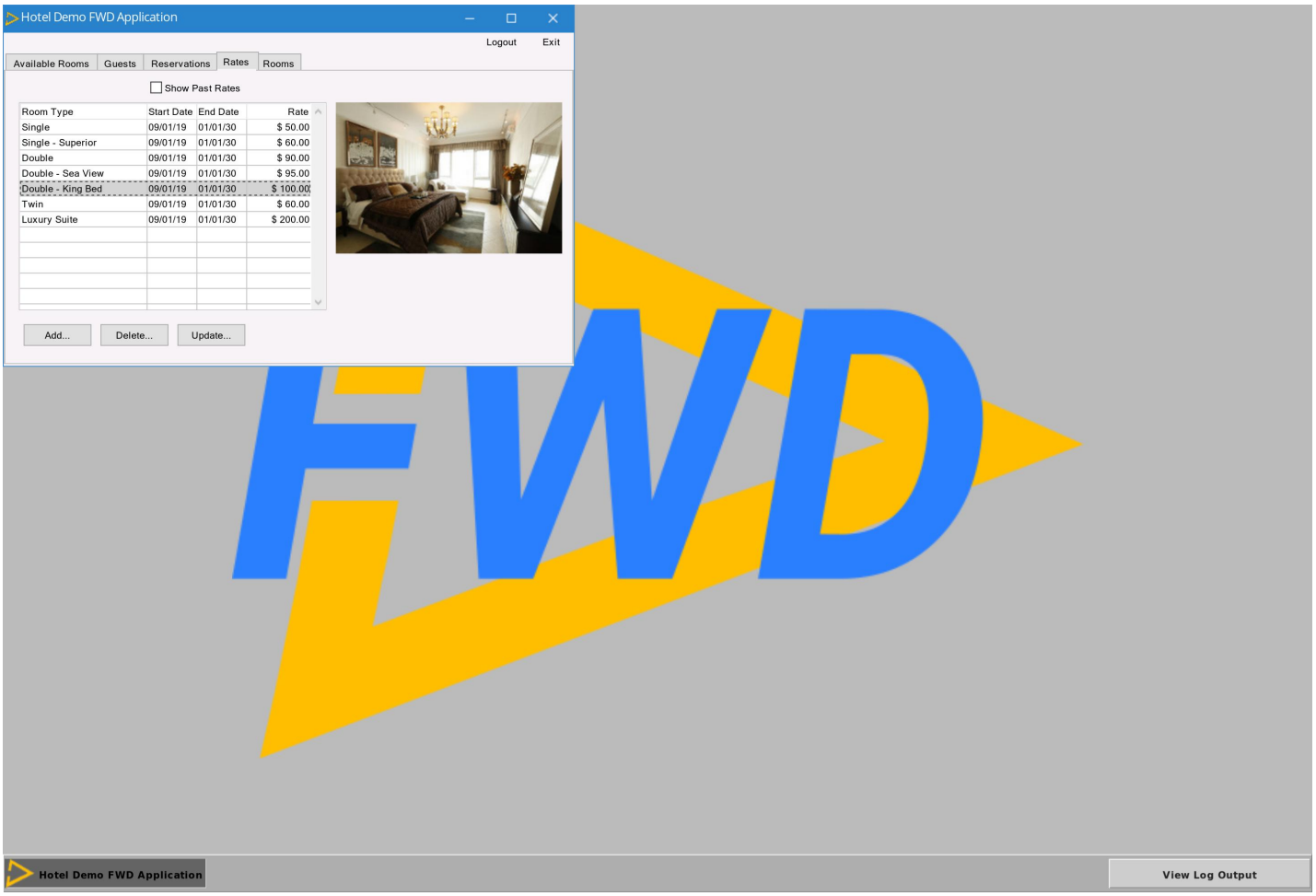
#33 - 11/06/2019 07:28 AM - Sergey Ivanovskiy

- File HotelDemoDPR2.png added

- File HotelDemo.png added

- File HotelDemoDPR2PrintScreen.png added

Please look at these 3 pictures. Two of them were done with Print Screen key and the last one was done with Take a Screen Shots of the viewport button of the Responsive Design Mode Firefox screen.



#34 - 11/06/2019 02:22 PM - Constantin Asofiei

Which one is with normal DPI? Also, please rebase 4135a when you can.

#35 - 11/07/2019 02:15 AM - Sergey Ivanovskiy

Planning to rebase now. The attached png files have comments so that DPR2 in their names means device pixel ratio 2 and it represents a screen shot for a high resolution screen. I used Firefox responsive design mode for testing. Two of them were produced by pressing Print Screen key and the last one was produced by Firefox itself by pressing on "Take a Screen Shots of the viewport" button.

HotelDemoDPR2PrintScreen.png - Print Screen key - high resolution screen (295 KB)

HotelDemo.png - Print Screen key - normal resolution (339 KB)

HotelDemoDPR2.png - Take a Screen Shots of the viewport (830 KB) - high resolution screen

#36 - 11/07/2019 03:22 AM - Sergey Ivanovskiy

4135a was updated up to rev. 11344 over trunc rev 11338.

#37 - 06/06/2020 12:15 PM - Greg Shah

From the customer:

> The file HotelDemoDPR2.png is looking good. Text looks sharp.
>
> As mentioned earlier in this threat the icons/images are not sharp, because they are scaled up. This the way to solve this in a normal web environment is to make a 200% version of an image and add @2x to the file name. E.g. icon.jpg (could be 32x32 px) and icon@2x.jpg (would be 64x64 px in this example). The HDPi browser will take the second version. We would need to generate a @2x image for all the used icons and images to make this work.

#38 - 08/11/2020 03:59 PM - Greg Shah

- Related to Bug #4849: Swing font size on laptop with HiDPI is too big added

Files

p2j.js.patch	1.36 KB	10/20/2019	Sergey Ivanovskiy
testDrawImage-1.html	5.48 KB	10/21/2019	Sergey Ivanovskiy
testDrawImage1.png	29.9 KB	10/21/2019	Sergey Ivanovskiy
DemoWidgetsDPIx2.png	123 KB	10/21/2019	Sergey Ivanovskiy
DemoWidgets1.png	143 KB	10/28/2019	Sergey Ivanovskiy
HotelDemoDPR2PrintScreen.png	295 KB	11/06/2019	Sergey Ivanovskiy
HotelDemo.png	339 KB	11/06/2019	Sergey Ivanovskiy
HotelDemoDPR2.png	830 KB	11/06/2019	Sergey Ivanovskiy