User Interface - Support #2672

GUI web client performance improvements

09/01/2015 02:00 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	40%
Category:		**Estimated time:**	0.00 hour
Target version:	Performance and Scalability Improvements		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to User Interface - Bug #2913: Improve window dragging in Web GUI client	**Feedback**	**12/03/2015**	
Related to User Interface - Bug #2887: web client flashing/drawing artifacts	**Closed**		
Related to User Interface - Feature #1811: implement the AJAX client driver	**Closed**	**11/01/2013**	**03/06/2014**
Related to User Interface - Feature #3246: reduce the amount of data being se...	**Closed**		

History

#1 - 09/01/2015 02:27 PM - Greg Shah

The following are ideas for performance improvements in the GUI web client.

1. Watching the log of drawing operations, it appears that several parts of a window are drawn multiple times during the same operation, for example on the initial display of a window. If this "overdrawing" can be eliminated, it should be a natural speed up for both the Swing and web clients.

2. Many sequences of drawing operations are very repetitive. For example, the parts of the window titlebar are usually static and the drawing is the same, over and over. We could create a CACHE_DRAWING_SEQUENCE drawing op to cache these sequences and associate them with an ID. Then the Java side could reference these cached sequences just by including a RENDER_CACHED op in a drawing ops batch (with the associated ID). There is no reason to send all the details down again unless something changes permanently, like if the window is resized. The reduction in network traffic could be significant. I suspect this would make things more responsive.

3. Closely related to the caching idea, we would want to enable caching and automatic replay of any drawing that looks like a kind of animation. For example, a button press causes the button to display pressed and then redisplay "popped back up". Handling this completely on the JS client side would make the client appear much more responsive.

4. Push as much web client logic processing into webworkers as is possible. Try to limit the UI thread processing to be very small and quick. If possible, only the rendering of the canvas pixels and the user-generated event handlers would be there. Even the event handlers would hand-off processing to webworkers as quickly as is possible. The biggest issue here is to ensure that we don't de-synchronize things that really do need to be synchronized.

5. Delegate mouse event processing to the web client as much as is possible. The idea is that for each mouse event that can actually change things on the Java side, the Java code would configure the web client with hit-testing rectangles and how they map into mouse events that need to be reported to the Java side. Mouse events that are not configured will not be sent, since they would just be ignored anyway. We already do some of this in the current implementation. This idea is intended to make a more complete solution, that even includes things like 4GL trigger awareness (if there are specific mouse event triggers that need to be honored, we need to configure them just like any other widget mouse requirement). Make sure that disabled widgets are removed from the hit testing.

6. Move Java code down to the JS side. This is the "ultimate" performance improvement and it can be used for many diverse cases. The idea is to refactor the Java code into a form such that the code to be moved down is isolated from code that should not move AND to remove any feature usage in that code which would not be easily converted to JS. Then we would use something like GWT or Scala.js during the build to create JS modules that can be easily leveraged in the hard coded JS code. One key area which is likely to be very useful here is FILL-IN editing (where the fill-in uses one of our DisplayFormat subclasses to manage the key entry in a format-specific manner).

#2 - 12/04/2015 07:45 AM - Greg Shah

We need to coordinate this work with #2913, which is related.

#3 - 12/10/2015 09:42 AM - Greg Shah

I'm adding some email discussion here for future reference, regarding font/text metrics and text editing:

On 11/27/2015 05:46 PM, Constantin Asofiei wrote:

> Sergey,
>
> Yes, text input (of any kind) will be problematic, in terms of measuring the text and drawing it. I think editable FILL-IN/EDITOR should be one of the first JS-optimized widgets.
> On 11/28/2015 12:18 AM, Serg Ivanovskiy wrote:
>
> > Greg,
> >
> > I only mean that if we type keys, the java side will ask about the height and the width of the typed text from the JS client. And we have a typed text as a key, thus the cache is not effective. At first I think it is enough to know the width and height for each font symbol to draw the user widget, because the width of the given text can be calculated as the upper bound that doesn't take into account kerning effect.
> >
> > Sergey
> > On 11/27/2015 09:09 PM, Greg Shah wrote:
> >
> > > Sergey,
> > >
> > > > 2) The text size requests from the java web client to JS client.
> > >
> > > Are you seeing any repeated requests? If so, then we might have an issue with our caching of results.
> > >
> > > Greg
> > > On 11/27/2015 11:55 AM, Serg Ivanovskiy wrote:
> > >
> > > > Greg,
> > > >
> > > > I am testing now, but it hasn't improved the performance yet. We have two known bottlenecks:
> > > > 1) The pixels operations
> > > > 2) The text size requests from the java web client to JS client.
> > > >
> > > > Sergey

#4 - 01/18/2016 03:40 PM - Constantin Asofiei

Some notes about my investigation/what I've done so far:

1. the most time-consuming drawing is related to drawing lines. I've tried to optimize this at some point, but to no avail. I'll give it another try next.
2. Firefox is a lot slower than Chrome, for some reason (but is good, as this gives us an easier way to profile)
3. I've added a MD5 hash at the driver level, on the java-side; there are two usages for now:
   - before sending the drawing ops to the JS side, if this has the same hash as the last batch sent to the JS-side, then drop the message, as it will duplicate the drawing
   - on the JS side, it uses a simple cache (for now, will switch to an expiry cache based on last access/usage count next) to determine if this batch was previously drawn, using the MD5 hash as key; if a cache hit is found, it will use the cached image and draw that instead (the non-drawn pixels of the cached image are transparent). For this, I've re-used the offscreenCanvas added by Sergey so that, after each batch drawing, it will save its content in the cache and clear itself - so next drawing will be on a cleared offscreen canvas. This reduces usage like button-press and other "static" drawing, but doesn't help text input. Another issue here is that this will work with only fully-bracketed drawing (i.e. explicit flush call currently used for StatusLine drawing will mess up the state) - but protection can be added and save the cache only if the translated origin/clipping count is 0 (i.e. batch has finished) when drawing completes
4. another issue is the disable/enable of OS events which is called each time control is switched to the server-side and received back by the P2J client. For the GUI app, even for the initial drawing, there were 1000s of these calls (resulted from widget attribute setter calls or 4GL drawing ops, from the 4GL business logic). We need to find a better approach for this... for now, I've disabled it, as it interfered with my profiling. In other words, currently each server-client trip requires 2 calls to the JS side for the OS event enable/disable, and this is costly.

#5 - 01/18/2016 11:32 PM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> Some notes about my investigation/what I've done so far:
>
> 1. the most time-consuming drawing is related to drawing lines. I've tried to optimize this at some point, but to no avail. I'll give it another try next.

I also tried to optimize line drawings without success. But this function function copyImage(img, data, imgData, width, height, imgDataOffset) introduced 1811t: p2j.canvas_renderer.js can be used to draw vertical and horizontal lines and filled rectangles by WidenPathRenderer to eliminate several calls of putImageData to one per drawing.
It can help to increase a bit these pixels operations.

#6 - 01/19/2016 06:27 AM - Constantin Asofiei

Greg, another optimization which will benefit the client in general: when setting a widget attribute, it should be a no-op if the widget's attribute value doesn't really change. For example, setting a widget's HIDDEN attr to FALSE when the HIDDEN is already FALSE results in an unnecessary trip to the client, repaint, etc. Same for size, location, sensitive (and maybe other) attributes.

#7 - 01/20/2016 09:42 AM - Constantin Asofiei

Greg/Sergey: I think I found a solution for drawing performance.

In p2j.canvas_renderer.js, we are drawing lines pixel-by-pixel, directly onto the canvas (off-screen or not, it doesn't matter). This includes calling a putImageData for each and every pixel - and from profiling this is the most time consuming function.

The alternative which:

- on firefox is ~60 times faster
- on chrome is ~6 times faster (but consider chrome's putImageData is already 20 times faster than firefox, so that's why the improvement is not the same)

is to use a separate transparent image (created via ctx.createImageData) and draw the individual pixels there. After drawing is finished, use a single putImageData to draw the result onto the real canvas.

A snapshot of the code I've used to prove this ("cr" is a CanvasRenderer instance), which fills a 500x500 rectangle by setting the color for each pixel:

```
var n = 500;

t1 = (new Date()).getTime();
// this is the optimized version which uses a single "putImageData", after all
// pixels are set.  firefox still has a "catch" here: it completes the loop
// in 100 * step time (i.e. 6s for a 60ms step), while chrome is 10 times faster
var imgData = cr.createImageData(n, n);
for (var j = 0; j <= 100; j++)
{
   for (var i = 0; i < imgData.data.length; i += 4)
   {
      imgData.data[i + 0] = (i * 3) % 255;
      imgData.data[i + 1] = (i * 2) % 255;
      imgData.data[i + 2] = i % 255;
      imgData.data[i + 3] = 255;
   }

   cr.ctx.putImageData(imgData, 0, 0);
   imgData.data.fill(255);
}

t2 = (new Date()).getTime();
console.log("done in: " + (t2 - t1));

// this is the unoptimized version (what P2J does currently): firefox is 20 times
// slower than chrome (~1200ms vs ~60ms)
t1 = (new Date()).getTime();
var imgData = cr.createImageData(1, 1);
for (var x = 0; x < n; x++)
{
   for (var y = 0; y < n; y ++)
   {
      imgData.data[0] = (x * 3) % 255;
      imgData.data[1] = (x * 2) % 255;
      imgData.data[2] = x % 255;
      imgData.data[3] = 255;

      cr.ctx.putImageData(imgData, x, y);
   }
}

t2 = (new Date()).getTime();
console.log("done in: " + (t2 - t1));
```

#8 - 01/20/2016 10:07 AM - Greg Shah

Great results!

I wonder if we can implement the createImageData() as a buffer for the entire window or does our drawing ops approach require that the imgData instance be implemented per drawing op?

#9 - 01/20/2016 10:17 AM - Constantin Asofiei

Greg Shah wrote:

> Great results!
>
> I wonder if we can implement the createImageData() as a buffer for the entire window or does our drawing ops approach require that the imgData instance be implemented per drawing op?

Actually, is enough to get the pixels directly from the image data of CanvasRenderer's canvas (so we don't have to create a new image data array). This test:

```
t1 = (new Date()).getTime();
var imgData = cr.ctx.getImageData(0, 0, n, n);
for (var j = 0; j <= 100; j++)
{
   for (var i = 0; i < imgData.data.length; i += 4)
   {
      imgData.data[i + 0] = (i * j * 3) % 255;
      imgData.data[i + 1] = (i * j * 2) % 255;
      imgData.data[i + 2] = (i * j) % 255;
      imgData.data[i + 3] = 255;
   }

   cr.ctx.putImageData(imgData, 0, 0);
   imgData.data.fill(255);
}

t2 = (new Date()).getTime();
console.log("3 done in: " + (t2 - t1));
```

gives almost identical times in both chrome and firefox.

The only catch is that, when setting the pixel, the translated origin needs to be accounted for.

#10 - 01/20/2016 10:30 AM - Greg Shah

Well done! Please go ahead with the change in 1811t.

#11 - 01/20/2016 10:40 AM - Greg Shah

1. Firefox is a lot slower than Chrome, for some reason (but is good, as this gives us an easier way to profile)

Do I understand correctly that the line drawing improvement eliminates this difference?

1. I've added a MD5 hash at the driver level, on the java-side; there are two usages for now:
   - before sending the drawing ops to the JS side, if this has the same hash as the last batch sent to the JS-side, then drop the message, as it will duplicate the drawing

Is there value in tracking a deeper pipeline of the recent drawing and then just sending the hash value (instead of the full low-level drawing ops list) when a match is detected? This savings in network transfer may add up. I imagine this would work well for things that animate like buttons, menus and so forth.

> Another issue here is that this will work with only fully-bracketed drawing (i.e. explicit flush call currently used for StatusLine drawing will mess up the state) - but protection can be added and save the cache only if the translated origin/clipping count is 0 (i.e. batch has finished) when drawing completes

I wonder if the hashing might work well at a per-widget level, instead of hashing on the entire batch. I would think that many full batches would contain some differences but still have a substantial amount of "static" or "static animation" (e.g. button presses) content.

1. another issue is the disable/enable of OS events which is called each time control is switched to the server-side and received back by the P2J client. For the GUI app, even for the initial drawing, there were 1000s of these calls (resulted from widget attribute setter calls or 4GL drawing ops, from the 4GL business logic). We need to find a better approach for this... for now, I've disabled it, as it interfered with my profiling. In other words, currently each server-client trip requires 2 calls to the JS side for the OS event enable/disable, and this is costly.

We previously found that we can't no-op the assignment on the P2J server side. I don't recall the exact cases, but I do recall that we had to make the trip down to the P2J client.

Are you saying that you think it is enough to go down to the P2J client, but not all the way to the JS side?

#12 - 01/20/2016 10:57 AM - Constantin Asofiei

Greg Shah wrote:

> 1. Firefox is a lot slower than Chrome, for some reason (but is good, as this gives us an easier way to profile)
>
> Do I understand correctly that the line drawing improvement eliminates this difference?

I can't confirm 100% that the GUI screens will be solved (as I don't have yet all the changes to test the drawing from canvas renderer), but considering the P2J profile results and standalone tests which show that putImageData is costly, it should solve it. I'm working on changing the canvas renderer now and will check some standalone drawing tests using P2J primitives, directly from JS. There is another complication here as canvas renderer uses getImageData too (but I think this can be replaced easily with getImageData function which just copies the pixels from the existing img data, not directly from the canvas).

> 1. I've added a MD5 hash at the driver level, on the java-side; there are two usages for now:
> - before sending the drawing ops to the JS side, if this has the same hash as the last batch sent to the JS-side, then drop the message, as it will duplicate the drawing
>
> Is there value in tracking a deeper pipeline of the recent drawing and then just sending the hash value (instead of the full low-level drawing ops list) when a match is detected? This savings in network transfer may add up. I imagine this would work well for things that animate like buttons, menus and so forth.

Yes, we can drop the entire drawing ops list, if the client side sees that a hash was already sent to the JS side (which should have cached it).

> Another issue here is that this will work with only fully-bracketed drawing (i.e. explicit flush call currently used for StatusLine drawing will mess up the state) - but protection can be added and save the cache only if the translated origin/clipping count is 0 (i.e. batch has finished) when drawing completes
>
> I wonder if the hashing might work well at a per-widget level, instead of hashing on the entire batch. I would think that many full batches would contain some differences but still have a substantial amount of "static" or "static animation" (e.g. button presses) content.

Yes, this would be the next level of improvement. For now, I want to stabilize the changes and finish the expiry cache on the JS side, and on the next round try to improve again.

> 1. another issue is the disable/enable of OS events which is called each time control is switched to the server-side and received back by the P2J client. For the GUI app, even for the initial drawing, there were 1000s of these calls (resulted from widget attribute setter calls or 4GL drawing ops, from the 4GL business logic). We need to find a better approach for this... for now, I've disabled it, as it interfered with my profiling. In other words, currently each server-client trip requires 2 calls to the JS side for the OS event enable/disable, and this is costly.
>
> We previously found that we can't no-op the assignment on the P2J server side. I don't recall the exact cases, but I do recall that we had to make the trip down to the P2J client.
>
> Are you saying that you think it is enough to go down to the P2J client, but not all the way to the JS side?

No, we can't avoid going to the JS side. I've disabled this for profiling reasons, to reduce "noise" and focus on drawing issues. But consider note 6 - if there are multiple server-client calls (which don't necessarily end up affecting the UI), then each server-client call will actually be a server-client-[js-side-enable-OS]-[client-does-work]-[js-side-disable-OS]-server chain - so we have two JS trips only for enabling/disabling OS events. And this is costly - I've seen this in the GUI project, even if a frame is already realized/placed/etc, the business logic sets (for example) the widget's location/size attributes, but the attribute value was not changed at all - so the server will call pushScreenDefinition, and client ends up reaching the JS side, even if it is not needed . We need to drop these cases directly from the server-side.

#13 - 01/20/2016 11:13 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> Greg/Sergey: I think I found a solution for drawing performance.
>
> In p2j.canvas_renderer.js, we are drawing lines pixel-by-pixel, directly onto the canvas (off-screen or not, it doesn't matter). This includes calling a putImageData for each and every pixel - and from profiling this is the most time consuming function.
>
> The alternative which:
>
> - on firefox is ~60 times faster
> - on chrome is ~6 times faster (but consider chrome's putImageData is already 20 times faster than firefox, so that's why the improvement is not the same)
>
> is to use a separate transparent image (created via ctx.createImageData) and draw the individual pixels there. After drawing is finished, use a single putImageData to draw the result onto the real canvas.
>
> A snapshot of the code I've used to prove this ("cr" is a CanvasRenderer instance), which fills a 500x500 rectangle by setting the color for each pixel:
> [...]

I only would like to notice that the offscreen canvas and the attached canvas should be different in performance with getImageData and putImageData. I think that the offscreen canvas can be optimized by the browser's engine, because the browser's screen renderer isn't involved for the offscreen canvas.

#14 - 01/20/2016 11:21 AM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> I only would like to notice that the offscreen canvas and the attached canvas should be different in performance with getImageData and putImageData. I think that the offscreen canvas can be optimized by the browser's engine, because the browser's screen renderer isn't involved for the offscreen canvas.

This might be a reason why Chrome is a lot faster than Firefox. But Firefox is slow regardless if the canvas is offscreen or not.

#15 - 01/20/2016 11:23 AM - Constantin Asofiei

Greg Shah wrote:

> 1. Firefox is a lot slower than Chrome, for some reason (but is good, as this gives us an easier way to profile)
>
> Do I understand correctly that the line drawing improvement eliminates this difference?

Standalone tests are promising: after the canvas renderer change, drawing rectangles using the P2J primitives results in similar times in both Firefox and Chrome.

#16 - 01/29/2016 07:23 AM - Sergey Ivanovskiy

I thought about the JS optimization to reduce numerous calls of puImageData and came to this plan:
1) Take the offscreen snapshot of the target clipping region with getImageData
2) Form (red, green, blue, alpha) array with the target primitive drawing, horizontal and vertical lines
3) Use function copyImage(img, data, imgData, width, height, imgDataOffset) from p2j.canvas_renderer.js. This function is well tested against all browsers.

#17 - 01/29/2016 11:16 AM - Constantin Asofiei

Greg, 1811t rev 10991 contains these:

1. the drawing cache (based on MD5 hashes). In a note you mentioned that we should not send the payload if the client already has a cached image for this MD5 hash. The problem here is that we should not cache ALL images (I'll add an expiry cache on JS side), so client side can't know if the cached image for this hash was dropped or not. But this can be solved if the JS side sends to the client-side the list of dropped hashes, so client-side can drop them too. I've disabled the flush() in status-line drawing - this corrupts the clip and translate counters, haven't found yet why.
2. misc fixes: border drawing for window must be done conditionally
3. a cleanup in CanvasRenderer - I've removed the ctx parameter for the APIs, as this class should use its saved context, and not an outside one.

Note about putImageData. In the end, even if I got past the translation/clipping isuses I was seeing, the improvement is not very visible. Before going into detail, some notes about the approach I used: use a separate ImageData instance where only line and rectangle-related APIs are drawing (actually, all APIs which ended up calling drawPixel()). When that API is finished, overlay the content of ImageData over the current canvas - the problem here is that a putImageData still needs to be involved (into another canvas), as we need to use drawImage() to overlay the current ImageData - as putImageData() is not aware of clipping or alpha.

Some other notes: drawing pixel-by-pixel in ImageData can not use "half pixels" - this was the biggest cause of the issues I was encountering. So coordinates will be miss-aligned, as we can't map the pixels in the image data with the pixels on the canvas. After removing the 0.5px translate, the screen looked good, but the performance was not improved, as I was still relying at some point on putImageData() to convert the in-memory ImageData into a Image object (using a ImageData-to-Canvas-to-Image approach). The alternative would be to base64 encode the Imagedata ourselves and not rely on canvas.toDataURL, to avoid the putImageData() - but I'm not sure of the performance of converting the byte-array->string->base64 conversion. Some improvements can be made if only the clipped rectangle is used. But in any case, drawing "half-pixels" into the ImageData still remains - so even if this proves OK, the half-pixels will still be problematic.

#18 - 01/31/2016 11:39 AM - Sergey Ivanovskiy

- File restore.png added

Constantin, please look at the attached picture of ./demo/simple_windows.p if the window's restore caption button is clicked, than its canvas displays the previous window state with a new one.

#19 - 02/01/2016 08:31 AM - Greg Shah

Is it still worth pursuing the putImageData() optimization? If the odds of success with this approach are diminishing, then we don't want to expend too much more time on it, unless you think we have reasonable odds of success.

Or is it time to consider moving widget-specific logic down to the client?

For example, what if a button generated the drawing ops for each of it's two states and then associated those with normal and pressed drawing conditions? Instead of leveraging a cache, we could draw based on JS events and only have to send the event notification to Java (but all drawing was directly handled in JS). If something changes in the button that causes the drawing ops to change, then the Java side would notify JS and provide the updates.

Another example: the content of scrolling viewports could be pre-drawn (or at least drawn more than what is visible) and then the scrolling logic could be moved to JS.

#20 - 02/03/2016 06:30 AM - Constantin Asofiei

1811t rev 11009 contains an expiry cache at the client-side (the MD5 producer). This is currently set to 1000 entries (should we make it configurable?). Also, it fixes a deadlock in WebClientProtocol.waitForResult, when waiting on receivedMessages - WebClientProtocol.lock should be used instead, as this is the socket lock.

Greg:

- about putImageData() - after the findings in note 17, I haven't worked on it any more and I will not, as is not a solution
- about the issue in note 6: we should really fix this, as lots of no-op attribute setter calls affect performance
- about what you note in note 19: I'm trying to find a way of caching the drawing operations for i.e. button press and use that (what I mentioned before as "intermediate hashes")

#21 - 02/03/2016 08:42 AM - Greg Shah

Thanks for the update.

> about the issue in note 6: we should really fix this, as lots of no-op attribute setter calls affect performance

Please go ahead with it.

> about what you note in note 19: I'm trying to find a way of caching the drawing operations for i.e. button press and use that (what I mentioned before as "intermediate hashes")

Good.

#22 - 02/08/2016 01:53 PM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> I thought about the JS optimization to reduce numerous calls of puImageData and came to this plan:
> 1) Take the offscreen snapshot of the target clipping region with getImageData
> 2) Form (red, green, blue, alpha) array with the target primitive drawing, horizontal and vertical lines
> 3) Use function copyImage(img, data, imgData, width, height, imgDataOffset) from p2j.canvas_renderer.js. This function is well tested against all browsers.

I found very interesting documentations about how to copy byte arrays using native API. It should decrease the overhead of the pixels operations:
Method 3: Transfer byte array by calling memcpy
https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes/Using_js-ctypes/Working_with_ArrayBuffers

#23 - 02/12/2016 04:34 AM - Sergey Ivanovskiy

- File performance_issue_1.txt added

#24 - 02/12/2016 05:42 AM - Sergey Ivanovskiy

- File deleted (performance_issue_1.txt)

#25 - 02/12/2016 05:43 AM - Sergey Ivanovskiy

- File performance_issue_1.txt added

Constantin, please look at this diff with the current 1811t. It works but the performance hasn't been increased yet. This method works only for horizontal and vertical lines, but the image for a sloped line should be more complicated.

#26 - 02/13/2016 05:05 PM - Sergey Ivanovskiy

- File performance_issue_2.txt added

Greg, Constantin, the committed revision 11020 reduces putImageData invocations for horizontal and vertical lines. Planing to reduce putImageData for CanvasRenderer.strokeLineSegment (WidenPathRenderer) in the case of vertical and horizontal lines. The note 22 opens optimization for the Firefox using the native OS functions, for an example memcpy if OS=Linux.

#27 - 02/15/2016 02:29 AM - Sergey Ivanovskiy

For vertical and horizontal lines the stroke that widens a line can be optimized to reduce putImageData, because the line from (x0, y0) to (x1, y1) is covered with this filled rectangle given by its left and top coordinates (x0 - (width>>1), y0 - (width>>1)) and its right and bottom coordinates (x0 - (width>>1) + width, y0 - (width>>1) + width).
Thus it can be optimized to fill the corresponding rgba array buffer and then to copy it to the background image data and finally to invoke putImageData. The stroke that makes dots also can be optimized to reduce putImageData. Constantin, does this plan interfere with your work? Could I proceed it?

#28 - 02/15/2016 04:40 PM - Sergey Ivanovskiy

- File performance_issue_3.txt added

Greg, please review the committed revision 11021. It reduced putImageData invocations for horizontal and vertical lines.

#29 - 02/15/2016 05:29 PM - Greg Shah

This text was written by Sergey in #2967 (note 21):

> In the case of the text editor we have numerous synchronous requests (calculate the width and the height of this text for me) from Java to JS and the proposed design adds a time to these requests trips.

It got me thinking. When the user is editing text in a fill-in or editor (or a browse cell/simple-mode combo-box that is a fill-in), it seems to me that we could be smart about eliminating these text sizing requests. By tracking the current line of text and the current caret position, when editing keystrokes occur, we could speculatively (and pre-emptively) measure the text sizes of the expected resulting text and send these from JS to Java along with the keystroke event itself. For most cases, we might be able to eliminate the down-call (Java to JS) if we have pre-calculated the correct result. I imagine it would have a potentially nice impact on performance.

Thoughts?

#30 - 02/15/2016 06:07 PM - Greg Shah

Code Review Task Branch 1811t Revisions 11020/11021

The changes in p2j.canvas_renderer.js are pretty straightforward. I think they are fine.

The changes in p2j.strokes.js are harder to verify. I've read the changes twice (quickly) so far and I'm going to have to read them again more carefully before I am fully sure it is OK.

The implementation is quite complex now and it is not obvious why certain choices are being made. It would help greatly if you would add some comments to the code to explain why key decisions are being made. For example, there are many places where we chose a different implementation based on some condition (e.g. browser is chrome or not, dx/dy is non-zero or not...). These are good locations for comments.

#31 - 02/16/2016 01:34 AM - Sergey Ivanovskiy

Greg Shah wrote:

> It got me thinking. When the user is editing text in a fill-in or editor (or a browse cell/simple-mode combo-box that is a fill-in), it seems to me that we could be smart about eliminating these text sizing requests. By tracking the current line of text and the current caret position, when editing keystrokes occur, we could speculatively (and pre-emptively) measure the text sizes of the expected resulting text and send these from JS to Java along with the keystroke event itself. For most cases, we might be able to eliminate the down-call (Java to JS) if we have pre-calculated the correct result. I imagine it would have a potentially nice impact on performance.
>
> Thoughts?

I would like to support this idea. It looks like if we do somehow two circles: the first one to draw the editor's without its data and the second to set text, then we can use your idea to improve the web editor's performance tremendously, because the first appearance of text components is its own drawing and then from JS client we send all required text's widths and caret's positions without synchronous calls from Java to the JS client and back!

#32 - 02/16/2016 11:48 AM - Sergey Ivanovskiy

Constantin, could you help? Does the Web Chui client support md5 hashing? It looks that p2j.socket.js is a common module and it expects that 0x81 messages have md5 hash keys. Thus the Web Chui doesn't work for 1811t.

#33 - 02/16/2016 12:02 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> Constantin, could you help? Does the Web Chui client support md5 hashing? It looks that p2j.socket.js is a common module and it expects that 0x81 messages have md5 hash keys. Thus the Web Chui doesn't work for 1811t.

I haven't checked the Web Chui client, so you are correct that it is broken. I'll move the hash related code into a client-specific function so that ChUI is not affected. I'm cleaning up/preparing the update and will commit it later today.

#34 - 02/16/2016 03:54 PM - Constantin Asofiei

1811t rev 11022 enhances the drawing cache to include intermediate widget states; currently, caching is used only for widgets which are fully being drawn. Also, it fixes the Web Chui client problem.

On the list remain the following, in order of priority:

1. fix the no-op attribute setters to not call the client-side (note 6). I want to work this in a different branch, as it is only server-side and should runtime test it separately.
2. expand the caching to include nested widgets (i.e. entire frame state) - infrastructure is there, but there are some issues if I enable this globally.
3. an existing issue in trunk: in Web client for test 454, after navigating through the bottom tabs and clicking between the browse widgets, there are some artifacts. I think some dimensions are not computed properly... haven't found the root cause yet
4. same, for test 454, in some cases the widget's size (what is actually being drawn, for a radio-set) doesn't match the rectangle from the posted PaintEvent - the clip (at the time of drawing) is smaller, and thus caching does not include it. Don't know yet if this is related to how test 454 works or is something in the P2J code.

#35 - 02/16/2016 04:30 PM - Constantin Asofiei

Sergey, about revision 11021; the changes look OK, but are you sure this gets us the same result as the previous implementation?

#36 - 02/16/2016 04:42 PM - Sergey Ivanovskiy

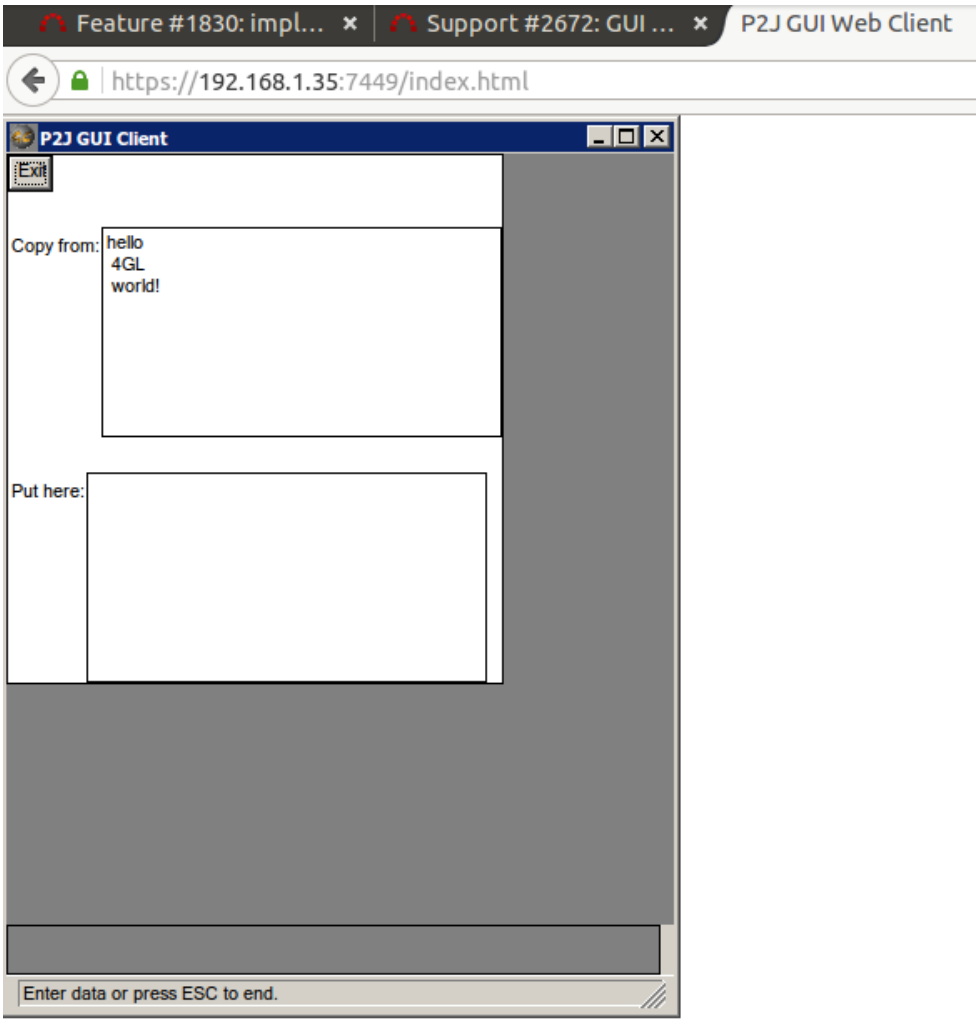
Constantin Asofiei wrote:

Sergey, about revision 11021; the changes look OK, but are you sure this gets us the same result as the previous implementation?

It should be the same as the previous implementation it has been tested with demo_widgets.p. The only difference is the DotsPathRenderer starts drawing dots for each new line, but the previous one continues the path of pixels. It can be fixed. Do you have new issues with my changes? Currently I have tested simple example with two editors against 11022 and found that the window icons are drawn without background and they look opaque. I am worried about the current changes because the image drawing is performed by the JS client.

#37 - 02/16/2016 04:50 PM - Sergey Ivanovskiy

- File icons_bg_issue.png added

The image background issue is 

#38 - 02/16/2016 04:52 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> The image background issue is ...

Thanks for pointing this, I'll look into it. I think some alpha info is being lost somewhere.

#39 - 02/16/2016 04:59 PM - Sergey Ivanovskiy

If we draw the image exactly like it comes from the Java, the result should be like we see, I have encountered it in the beginning of this drawing icons task. Constantin, are there urgent reasons to change the code to 11022 if it adds new issues or you are planing to stabilize it sooner? There are new artifacts (some drawing lines) that appear if we do clicking on controls, I am only worried.

#40 - 02/16/2016 05:00 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> It should be the same as the previous implementation it has been tested with demo_widgets.p. The only difference is the DotsPathRenderer starts drawing dots for each new line, but the previous one continues the path of pixels. It can be fixed.

What I'm worried is about the case in this image: https://proj.goldencode.com/redmine/attachments/download/4108 from #2564-149 See how the dotted rectangle for the selected row is contiguous at the corners. This is not specific only to browse, is specific to any other case where there is a dotted rectangle to mark a selection. Does this still work the same? Also, please check the pressed/unpressed state of the window close button: this is another part where we had problems in duplicating the "X".

> Do you have new issues with my changes?

No.

#41 - 02/16/2016 05:05 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> If we draw the image exactly like it comes from the Java, the result should be like we see, I have encountered it in the beginning of this drawing icons task. Constantin, are there urgent reasons to change the code to 11022 if it adds new issues or you are planing to stabilize it sooner? There are new artifacts (some drawing lines) that appear if we do clicking on controls, I am only worried.

Can you provide an example where you saw this? I've seen something which might be the same in test 454, but I thought it was existing in trunk...

#42 - 02/16/2016 05:38 PM - Sergey Ivanovskiy

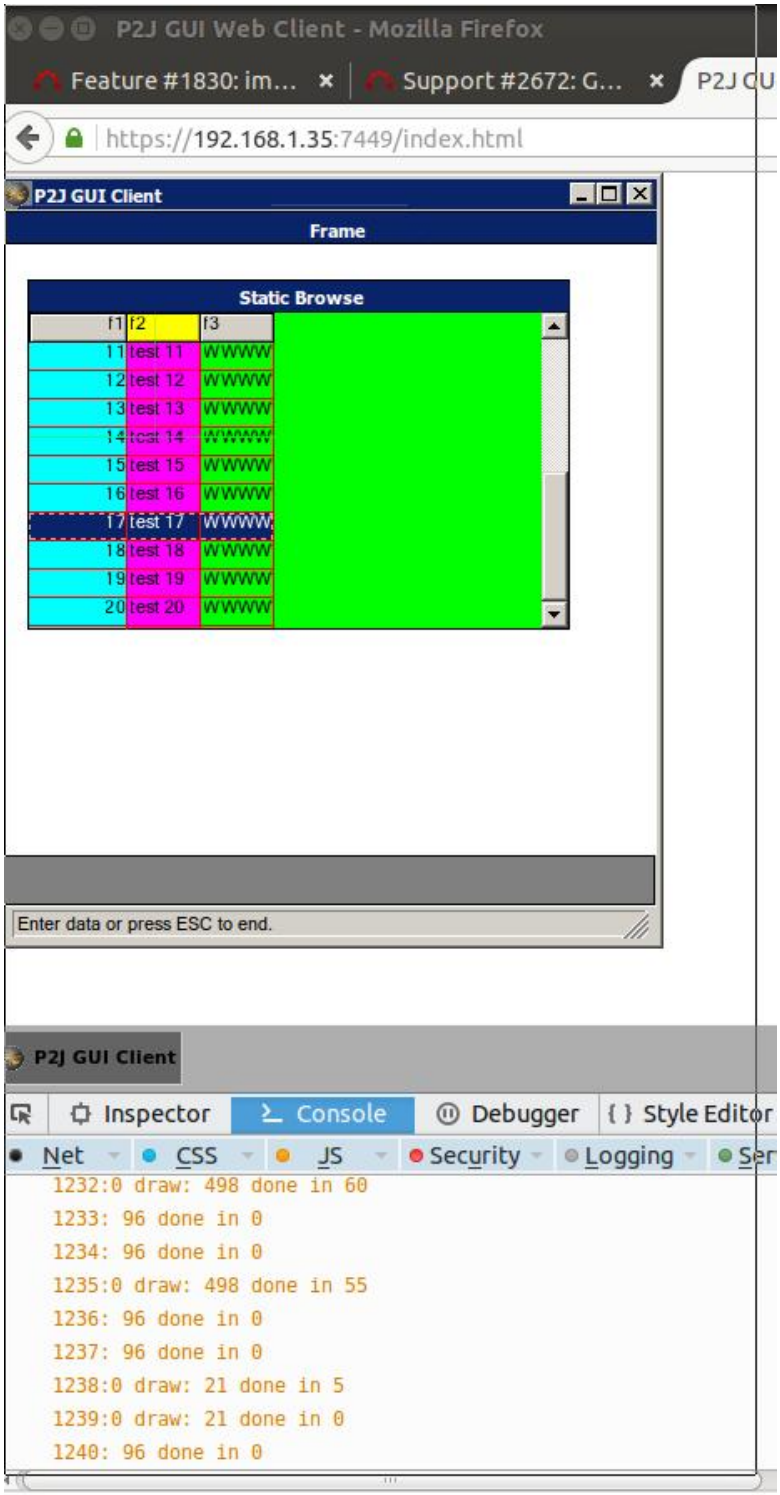
- File browse.png added

It looks the same for ./browse/gui/browse-gui-stat1.p, I am planning to check the dots drawing and to do it the same as in the previous version. The artifacts appear once at the start after new version is compiled and they are not reproduced now except if we click on the Restore/Maximize caption button after resizing the window we get the old image with new one. The same problem was in the previous version also.

#43 - 02/16/2016 05:53 PM - Sergey Ivanovskiy

- File browse_artifacts_in_title.png added

The difficult problems in ./browse/gui/browse-gui-stat1.p are



It happens after clicking on the Close caption button, but it is not reproduced each time after clicking.

#44 - 02/17/2016 12:43 AM - Sergey Ivanovskiy

It looks that the new caching introduced by 11022 is more slower than the previous version 11021.
The reported times for draw became greater than in the previous version against ./demo/demo_widgets.p:

```
172: 8c done in 20 p2j.socket.js:1038:13
restack request p2j.screen.js:2320:7
1 p2j.screen.js:2323:10
restack p2j.screen.js:2371:7
Object { id: 1, win: Object } p2j.screen.js:2374:10
173: 95 done in 24 p2j.socket.js:1038:13
174: 9c done in 17 p2j.socket.js:1038:13
175:0 draw: 6892 done in 534 p2j.socket.js:487:19
176: 8f done in 0 p2j.socket.js:1038:13
177: 8e done in 1 p2j.socket.js:1038:13
178:0 draw: 792 done in 170 p2j.socket.js:487:19
179: 98 done in 0 p2j.socket.js:1038:13
180: 98 done in 0 p2j.socket.js:1038:13
181: 8f done in 1 p2j.socket.js:1038:13
182: 8e done in 1 p2j.socket.js:1038:13
183:0 draw: 930 done in 194 p2j.socket.js:487:19
184: 98 done in 0 p2j.socket.js:1038:13
185: 98 done in 0 p2j.socket.js:1038:13
186: 8f done in 0 p2j.socket.js:1038:13
187: 8e done in 1 p2j.socket.js:1038:13
188:0 draw: 3256 done in 302 p2j.socket.js:487:19
189: 98 done in 0 p2j.socket.js:1038:13
190: 98 done in 0 p2j.socket.js:1038:13
191: 8f done in 0 p2j.socket.js:1038:13
192: 8e done in 1 p2j.socket.js:1038:13
193:0 draw: 1746 done in 315 p2j.socket.js:487:19
194: 98 done in 0 p2j.socket.js:1038:13
195: 98 done in 0 p2j.socket.js:1038:13
196: 8f done in 0 p2j.socket.js:1038:13
197: 8e done in 2 p2j.socket.js:1038:13
Using global cached draw be5e57dfcdae47a636b1e13ec770fd87 posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
Using global cached draw e92250684a0ae8021b7c67a87cb04464 posted at coordinates x:387;y:500 p2j.canvas_rendere
r.js:340:4
198:0 draw: 1186 done in 183 p2j.socket.js:487:19
199: 98 done in 0 p2j.socket.js:1038:13
200: 98 done in 0 p2j.socket.js:1038:13
201: 8f done in 0 p2j.socket.js:1038:13
202: 8e done in 2 p2j.socket.js:1038:13
Using global cached draw e344005b2b9640256bfe6190625708cf posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
203:0 draw: 1011 done in 194 p2j.socket.js:487:19
204: 98 done in 0 p2j.socket.js:1038:13
205: 98 done in 0 p2j.socket.js:1038:13
206: 8f done in 0 p2j.socket.js:1038:13
207: 8e done in 1 p2j.socket.js:1038:13
Using global cached draw e344005b2b9640256bfe6190625708cf posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
208:0 draw: 1011 done in 163 p2j.socket.js:487:19
209: 98 done in 0 p2j.socket.js:1038:13
210: 98 done in 0 p2j.socket.js:1038:13
211: 8f done in 0 p2j.socket.js:1038:13
212: 8e done in 2 p2j.socket.js:1038:13
Using global cached draw e344005b2b9640256bfe6190625708cf posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
213:0 draw: 1011 done in 189 p2j.socket.js:487:19
214: 98 done in 0 p2j.socket.js:1038:13
215: 98 done in 0 p2j.socket.js:1038:13
216: 8f done in 0 p2j.socket.js:1038:13
217: 8e done in 6 p2j.socket.js:1038:13
Using global cached draw e344005b2b9640256bfe6190625708cf posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
218:0 draw: 1011 done in 191 p2j.socket.js:487:19
219: 98 done in 0 p2j.socket.js:1038:13
220: 98 done in 0 p2j.socket.js:1038:13
221: 8f done in 0 p2j.socket.js:1038:13
222: 8e done in 1 p2j.socket.js:1038:13
Using global cached draw e344005b2b9640256bfe6190625708cf posted at coordinates x:387;y:486 p2j.canvas_rendere
r.js:340:4
```

```
223:0 draw: 1013 done in 207 p2j.socket.js:487:19
224: 98 done in 0 p2j.socket.js:1038:13
225: 98 done in 0 p2j.socket.js:1038:13
226: 98 done in 0 p2j.socket.js:1038:13
227: 98 done in 0 p2j.socket.js:1038:13
228:0 draw: 1299 done in 258 p2j.socket.js:487:19
229: 98 done in 0 p2j.socket.js:1038:13
```

#45 - 02/17/2016 12:49 AM - Sergey Ivanovskiy

I think that getImageData is also an expensive operation.

#46 - 02/17/2016 01:45 AM - Sergey Ivanovskiy

I would like to notice the main problem related to the Web client performance is synchronous requests from the Java side to the JS client and back to calculate the width and the height of the given text for the Java side.

#47 - 02/17/2016 08:11 AM - Constantin Asofiei

Constantin Asofiei wrote:

> Sergey Ivanovskiy wrote:
>
> > The image background issue is ...
>
> Thanks for pointing this, I'll look into it. I think some alpha info is being lost somewhere.

Actually the reason is that when the cached image is captured, the background is "disabled". And as the image is transparent, next time it is drawn it will use the cache, which is not correct. The solution, as we can only capture rectangular areas from the canvas (and we actually don't know the shape of the image and which parts are transparent or not) is to exclude the image widget from the cache.

#48 - 02/17/2016 11:02 AM - Sergey Ivanovskiy

Constantin, I found the performance issue with ./demo/movie-ratings-dynamic.p (com.goldencode.testcases.demo.MovieRatingsDynamic.execute). The message "script is busy" appeared. Have tested it for revisions: 11021, 11020, 11019, it looks that 11019 is the best and 11020 is more slow. Thus the getImageData putImageData optimization doesn't work and I would like to revert to the previous design with putImageData for each pixel in the stroked line.

#49 - 02/17/2016 11:43 AM - Sergey Ivanovskiy

Before 11021 changes these primitives: drawRect, drawRoundRect, draw3DRect draws the stroked lines only if the current stroke is the default stroke with 1 pixel width, but in 11021 these primitives draw the stroked lines with the current stroke that can have any style. I don't know it is correct to stroke lines only if the current stroke is the default stroke for these drawRect, drawRoundRect, draw3DRect functions?

#50 - 02/17/2016 11:50 AM - Sergey Ivanovskiy

Also it is my fault that I don't check the performance for these two cases:
1) draw vertical|horizontal line by pixels using putImageData
2) getImageData for vertical|horizontal line image and then fill it with color, then put it on the screen with putImageData.
I am planing to check it with script test and will provide the JS test and its result here. Depending on the result I will roll back 11021 changes or not.
Greg, Constantin, do you agree?

#51 - 02/17/2016 01:34 PM - Greg Shah

> I am planing to check it with script test and will provide the JS test and its result here. Depending on the result I will roll back 11021 changes or not.

> Greg, Constantin, do you agree?

Yes, the plan seems reasonable.

#52 - 02/17/2016 02:48 PM - Sergey Ivanovskiy

- File LineStrokeTest2.html added

I did this simple test to draw 160 vertical lines of 400 pixel width

for DefaultPathRenderer with the default style

to draw lines as images render.strokeLine(...):

```
Average X-lines of 400 pixel width draw time is 8.15 LineStrokeTest2.html:110:13
All time to draw 160 X-lines of 400 pixel width draw time is 1304 LineStrokeTest2.html:111:13
```

to draw lines as pixels render.strokePoint(...):

```
Average X-lines of 400 pixel width draw time is 7.94375 LineStrokeTest2.html:131:13
All time to draw 160 X-lines of 400 pixel width draw time is 1271 LineStrokeTest2.html:132:13
```

for DotsPathRenderer with the dots style

to draw lines as images render.strokeLine(...):

```
Average X-lines of 400 pixel width draw time is 8.325 LineStrokeTest2.html:110:13
All time to draw 160 X-lines of 400 pixel width draw time is 1332 LineStrokeTest2.html:111:13
```

to draw lines as pixels render.strokePoint(...):

```
Average X-lines of 400 pixel width draw time is 3.8625 LineStrokeTest2.html:131:13
All time to draw 160 X-lines of 400 pixel width draw time is 618 LineStrokeTest2.html:132:13
```

for WidenPathRenderer with the style width equals 2 pixels
to draw lines as images render.strokeLine(...)

```
Average X-lines of 400 pixel width draw time is 8.06875 LineStrokeTest2.html:110:13
All time to draw 160 X-lines of 400 pixel width draw time is 1291 LineStrokeTest2.html:111:13
```

to draw lines as pixels render.strokePoint(...)

```
Average X-lines of 400 pixel width draw time is 8.0875 LineStrokeTest2.html:131:13
All time to draw 160 X-lines of 400 pixel width draw time is 1294 LineStrokeTest2.html:132:13
```

This test proves that only for WidenPathRenderer that is responsible to widen lines there is a slight improvement if we draw vertical lines as images, but for dots style and for the default style having the solid line width equals 1 pixel the drawing by pixels is preferred.
If you want to check and to modify the LineStrokeTest2.html test, you need to create client and common directories within the root directory containing this html file and place the p2j current scripts in these directories according to this rule: common modules to common directory and web gui modules to client directory

#53 - 02/17/2016 02:51 PM - Sergey Ivanovskiy

Greg, Do you agree if I will roll back 11021 changes according to this test?

#54 - 02/17/2016 03:31 PM - Greg Shah

Greg, Do you agree if I will roll back 11021 changes according to this test?

Yes, roll it back. Even for the WidenPathRenderer, it is essentially no better and the others are definitely worse.

#55 - 02/17/2016 03:49 PM - Sergey Ivanovskiy

Committed revision 11023 removes 11020 and 11021 changes.

#56 - 02/17/2016 04:03 PM - Sergey Ivanovskiy

Greg, I change some method and gives improvement
look at this instead of get we can use create in some cases for WidenPathRenderer

```
      this.strokeLine = function(x1, y1, x2, y2)
      {
         var lineWidth  = x2 - x1 + width;
         var lineHeight = y2 - y1 + width;
         var x0 = x1 - (width >> 1);
         var y0 = y1 - (width >> 1);
         var screenCapture = canvasRenderer.ctx.createImageData(lineWidth, lineHeight);
         var data = screenCapture.data;
         var length = 4 * lineWidth * lineHeight;

//          //var imageBuffer     = new ArrayBuffer(length);
//          var imageBufferView = new Uint8ClampedArray(data.buffer);
//          var imageUint32View = new Uint32Array(data.buffer);
//          // TODO: IE doesn't support fill function, but it can be used for others: Firefox and Chrome
//          imageBufferView[0] = strokeColor[0];
//          imageBufferView[1] = strokeColor[1];
//          imageBufferView[2] = strokeColor[2];
//          imageBufferView[3] = 255;
//          var fillValue = imageUint32View[0];
//          imageUint32View.fill(fillValue);

         for (var i = 0; i < length; i += 4)
         {
            data[i]     = strokeColor[0];
            data[i + 1] = strokeColor[1];
            data[i + 2] = strokeColor[2];
            data[i + 3] = 255;
//             imageBufferView[i]     = strokeColor[0];
//             imageBufferView[i + 1] = strokeColor[1];
//             imageBufferView[i + 2] = strokeColor[2];
//             imageBufferView[i + 3] = 255;
         }
         // In Chrome screenCapture image size can be different from lineWidth x lineHeight
//          if (p2j.isChrome)
//          {
//             canvasRenderer.copyImage(screenCapture, data, imageBufferView, lineWidth, lineHeight, 0);
//          }
//          else
//          {
//             // TODO: this can be optimized in Firefox
//             data.set(imageBufferView, 0);
//          }
         canvasRenderer.putImageData(screenCapture, x0, y0);
      }
```

and the test results are this one

```
Average X-lines of 400 pixel width draw time is 0.1125 LineStrokeTest2.html:110:13
All time to draw 160 X-lines of 400 pixel width draw time is 18 LineStrokeTest2.html:111:13
```

against this one

```
Average X-lines of 400 pixel width draw time is 8.18125 LineStrokeTest2.html:131:13
All time to draw 160 X-lines of 400 pixel width draw time is 1309 LineStrokeTest2.html:132:13
```

Sorry, but we can improve the performance of WidenPathRenderer.

#57 - 02/17/2016 04:11 PM - Sergey Ivanovskiy

Greg, let me try with the 11021 design taking into account the tests result about the pixels operations.

#58 - 02/17/2016 04:43 PM - Sergey Ivanovskiy

- File performance_issue_4.txt added

Sergey Ivanovskiy wrote:

> Greg, let me try with the 11021 design taking into account the tests result about the pixels operations.

Did it, the performance is improved. ./demo/movie-ratings-dynamic.p (com.goldencode.testcases.demo.MovieRatingsDynamic.execute) is flying now. Greg, please review the committed revision 11024.

#59 - 02/17/2016 05:44 PM - Greg Shah

Very cool results! Great job!

I've taken a quick look. Please add the comments I requested in note 30. Then I will do a more careful review.

Does this only improve the WidenPathRenderer or is there some benefit in the other cases?

#60 - 02/18/2016 06:25 AM - Sergey Ivanovskiy

It should improve drawings of lines(horizontal and vertical) having 1 pixel width and lines having 2 or more pixels width. The dots stroke style requires a getImageData invocation to draw a line as an image. I will check if it can be improved using one createImageData and one getImageData invocations. But I think that var image = getImageData(...) returns a wrapped object with lazily fed properties. It looks that the data is retrieved at the time of the member access image.data[index]. Committed revision 11025 added new and fixed old comments.

#61 - 02/19/2016 12:52 AM - Sergey Ivanovskiy

> Greg Shah wrote:
>
> > It got me thinking. When the user is editing text in a fill-in or editor (or a browse cell/simple-mode combo-box that is a fill-in), it seems to me that we could be smart about eliminating these text sizing requests. By tracking the current line of text and the current caret position, when editing keystrokes occur, we could speculatively (and pre-emptively) measure the text sizes of the expected resulting text and send these from JS to Java along with the keystroke event itself. For most cases, we might be able to eliminate the down-call (Java to JS) if we have pre-calculated the correct result. I imagine it would have a potentially nice impact on performance.
> >
> > Thoughts?

Constantin, what do you think about synchronous requests from the Java side to the JS client? Is it possible to change the logic in the editor that the JS client draws the editor content empty at its first appearance? As soon as the JS client gets the asynchronous update request (so the Java side doesn't wait the JS client) with the data for the editor, on this request the JS client calculates the data dimensions and sends it back to the Java side. The Java side sets the text and uses the calculated dimensions to format the text and then performs drawings on the JS client. It looks more simple than the editor component on the JS client. But I didn't investigate how to change the editor logic so this plan can be implemented. Are you planning to work on this?

#62 - 02/19/2016 08:13 AM - Constantin Asofiei

About the editing and showing data on the editor/fill-in:

1. for editor, let the driver parse the text and split it into lines. This way there will be only a single call the the driver, which will measure it and split it. Currently, this is done on a word-by-word basis. (what Editor.parseContent does).
2. when user moves the caret on a line, we could let JS send all intermediate text widths, for the current line, from first char to the last. This will make moving caret through the line more simple, and we will not rely on JS trips to measure texts on each move.
3. for editor/fillin: as Greg mentioned, we can compute the new line width and send this together with the keystroke. But again, this may be problematic: the line content might have changed if the key has a trigger attached, which changes the editor content. Instead, what we can do is use an approach where we measure text on Java side (using the deleted/inserted character width stored in the cached FontMetricsHelper instances) and, if the new line width doesn't trigger changes in other lines (i.e. line will not be split/merged with other lines), just use the char. To ensure we compute correctly the edge case when the char triggers line split, we can substract from the current line's text width the width of W char (a wide char), letting the JS side measure the text, to ensure the text can be accommodated by the line or not.
4. another improvement can be this: currently, when editing, regardless if only the content in a single line changes, the full editor is being drawn. We can make this so that only the affected line(s) are drawn - this will limit the drawing and should make the editing faster for large texts. This also needs to keep in mind the scrollbar.
5. about using async request for editor: 4GL is synchronous by nature, and I don't like changing this. At the time the JS async request sends the response back to the Java side, we don't know where the business logic will be. The editor content may not even be the same at the time this is processed by the client; more, if there are multiple async requests at the same time, we will need to ensure these are processed in the proper order, as if content changes, we will not be able to ensure that what is being drawn is what the editor currently contains.

#63 - 02/19/2016 08:58 AM - Constantin Asofiei

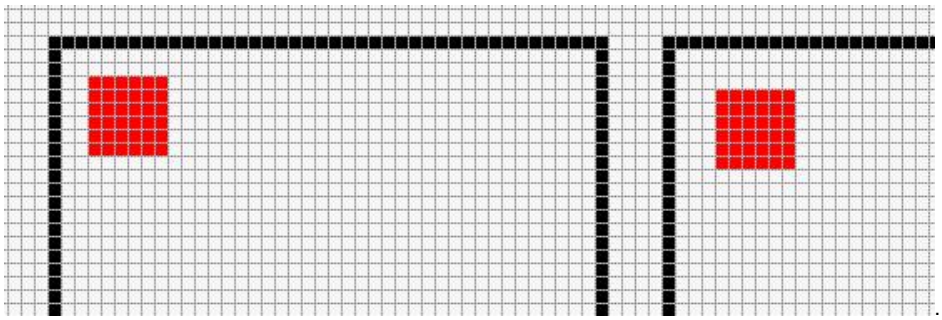
- File js-clip.png added

- File jsclip.html added

Constantin Asofiei wrote:

> 3. an existing issue in trunk: in Web client for test 454, after navigating through the bottom tabs and clicking between the browse widgets, there are some artifacts. I think some dimensions are not computed properly... haven't found the root cause yet

What I'm seeing here is caused by the this.translate(0.5, 0.5); call in p2j.canvas_renderer.js. If I remove this call,then the artifact is gone. I think the clip in JS doesn't work properly with sub-pixels. See the attached test (also here: https://jsfiddle.net/o1vx5xzv/): the left is without translate, the right is with translate(0.5, 0.5).



See how the right-side image is drawn on pixel 4, while the left-side is drawn on pixel 3. Sergey, any idea if we can fix this?

#64 - 02/19/2016 09:29 AM - Sergey Ivanovskiy

Constantin, please could you explain it more thoroughly.

The first script should intersect this rectangle(0, 0, 8, 8) and this clipping region(2, 2, 10, 10), isn't it?

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
// Clip a rectangular area
ctx.rect(2, 2, 10, 10);
ctx.clip();
// Draw red rectangle after clip()
ctx.fillStyle = "red";
ctx.fillRect(0, 0, 8, 8);
```

```
</script>
```

The second script should translate the origin to (0.5, 0.5) and intersect this rectangle(0, 0, 8.5, 8.5) and this clipping region(2, 2, 10, 10), isn't it?

```
<script>
var c = document.getElementById("myCanvas2");
var ctx = c.getContext("2d");
ctx.translate(0.5, 0.5);
// Clip a rectangular area
ctx.rect(2, 2, 10, 10);
ctx.clip();
// Draw red rectangle after clip()
ctx.fillStyle = "red";
ctx.fillRect(0, 0, 8.5, 8.5);
</script>
```

What results should be correct? What do you expect in dimensions? I used Firefox to test your example.

#65 - 02/19/2016 09:43 AM - Sergey Ivanovskiy

I increased the sizes of regions to watch the picture and the second red squared rectangle is antialiased. It has smooth edges.

#66 - 02/19/2016 09:51 AM - Sergey Ivanovskiy

If you removed the translate(0.5, 0.5) call, then some artifacts disappeared, didn't they? Now I have no ideas because don't understand the root cause. Please could you explain it? Sometimes trying to explain the issue helps to solve it.

#67 - 02/19/2016 09:57 AM - Sergey Ivanovskiy

In Chrome I found this issue if we get var image = ctx.getImageData(0.5,0.5, 10, 1), then the returned image dimensions are changed and don't equal to 10 x 1.

#68 - 02/19/2016 10:07 AM - Sergey Ivanovskiy

Another argument can be if we test our system against hello.p, then are these artifacts present?

#69 - 02/19/2016 10:40 AM - Constantin Asofiei

Sergey, I've seen this only in a scenario for test 454 and also when the window popup-menu is shown then cleared (any test can be used for this). Removing the translate solves both issues.

I don't have a full picture yet to understand the root cause; what surprised me was that I was expected for the fillRect in the example I posted to draw at pixel 2 (or some anti-aliasing at row 2/3), not directly at 3.

#70 - 02/19/2016 10:54 AM - Sergey Ivanovskiy

Ok, we can create a task to fix artifacts that appear in the case of popup-menu with the most simple 4GL test case program.

#71 - 02/19/2016 10:55 AM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> Ok, we can create a task to fix artifacts that appear in the case of popup-menu with the most simple 4GL test case program.

I'll make a task and post the relevant screenshots there.

#72 - 02/19/2016 01:09 PM - Constantin Asofiei

1811t rev 11026 fixes:

1. the window icon - we must not cache image widgets, as they are not rectangular and can be transparent.
2. fixed AbstractWidget.clipRectangle (the bottom/right coordinates were computed incorrect)
3. fixed mouse click/press in disabled radio-set
4. added a way to view clip rectangles in web client

#73 - 02/19/2016 01:10 PM - Constantin Asofiei

Constantin Asofiei wrote:

> 2. fixed AbstractWidget.clipRectangle (the bottom/right coordinates were computed incorrect)

This was the root cause for this item mentioned in note 34:

> Same, for test 454, in some cases the widget's size (what is actually being drawn, for a radio-set) doesn't match the rectangle from the posted PaintEvent - the clip (at the time of drawing) is smaller, and thus caching does not include it. Don't know yet if this is related to how test 454 works or is something in the P2J code.

#74 - 02/23/2016 11:44 AM - Greg Shah

Custom cursor support in JS is currently implemented without any caching of the resulting prepared cursor. In other words, it is recreated on the fly when it is assigned. There is an alternate approach that is likely to be faster. See #2565-192 and #2565-229 for details.

#75 - 02/28/2016 08:56 AM - Sergey Ivanovskiy

Constantin, according to the test getImageData and putImageData are expensive operations. Did you compare the web performance in the case if we wouldn't use the drawing cache?

#76 - 02/29/2016 12:59 AM - Sergey Ivanovskiy

Some remarks to discuss are
There are differences in the web performance depending on the used strategy to do caching.
1) Use the drawing cache and make a decision to fill it on the web java side like it is done for icons (the static cache is used to reduce the data required to send to the client)
2) Use the drawing cache and make a decision to fill it using interactions with the JS client (the dynamic cache is used for * ?)

#77 - 03/04/2016 12:25 PM - Constantin Asofiei

1811t rev 11053 compacts the "enable OS events" calls for the web driver, in a single call: all window IDs are passed to the websocket and processed in batch.

#78 - 03/04/2016 12:40 PM - Constantin Asofiei

In a BROWSE widget, for a single mouse click in a row, there are 6 client-server trips for just this row selection. For other cases (like query repositioning or close), there are also 3-4 trips (to get current row, rows). These are very expensive. For cases like ServerExports or ClientExports APIs which need only get some server-side state or data (and are non-blocking and don't affect UI), I think we should not execute the tk.enableOSEvents.

2672a revision 10980 adds no-op for attribute setters which allow this, but the impact is not sufficient in improving performance. This can be either merged into trunk or into 1811t.

#79 - 03/04/2016 01:53 PM - Greg Shah

> For cases like ServerExports or ClientExports APIs which need only get some server-side state or data (and are non-blocking and don't affect UI), I think we should not execute the tk.enableOSEvents.

OK, go ahead with this.

> 2672a revision 10980 adds no-op for attribute setters which allow this, but the impact is not sufficient in improving performance. This can be either merged into trunk or into 1811t.

Are you saying that there is no advantage to implementing the no-op attribute setters?

#80 - 03/04/2016 03:02 PM - Constantin Asofiei

Greg Shah wrote:

> 2672a revision 10980 adds no-op for attribute setters which allow this, but the impact is not sufficient in improving performance. This can be either merged into trunk or into 1811t.

Are you saying that there is no advantage to implementing the no-op attribute setters?

The improvement is ~10-15%, in numbers of JS-side calls for enableOsEvents (plus all the pushScreenDefinition overhead is avoided). But it really depends on business logic, other projects may be better improved by this.

#81 - 03/04/2016 03:56 PM - Greg Shah

Code Review Task Branch 2672a Revision 10980

I'm OK with the changes.

My only question: what was the process that you used to determine these attributes were safe to turn into no-ops?

#82 - 03/04/2016 03:59 PM - Constantin Asofiei

Greg Shah wrote:

> My only question: what was the process that you used to determine these attributes were safe to turn into no-ops?

I have tests which check attribute states before and after a no-op setter for an attribute is called, in a frame. There are some exceptions (for example, just accessing the SCREEN-VALUE attribute sets the MODIFIED to TRUE, for the widget). I will document them here (and in the P2J code, where the attributes are defined).

#83 - 03/04/2016 04:10 PM - Constantin Asofiei

Greg Shah wrote:

> For cases like ServerExports or ClientExports APIs which need only get some server-side state or data (and are non-blocking and don't affect UI), I think we should not execute the tk.enableOSEvents.

> OK, go ahead with this.

I think is better a reverse approach: mark only the APIs which actually affect the UI. Because there are other static network server implementations (like FileSystemDaemon) which don't even need the StateSynchronizer.

So, what about this: only APIs marked with i.e. a StateSync annotation will actually use this feature (APIs like ClientExports.hide, display, etc). Otherwise, we are passing back and forth serialized ClientState and ServerState instances with each and every network message between the client and the server, regardless if the API call was even able to change state in these objects.

#84 - 03/04/2016 04:23 PM - Greg Shah

> I think is better a reverse approach: mark only the APIs which actually affect the UI. Because there are other static network server implementations (like FileSystemDaemon) which don't even need the StateSynchronizer.
>
> So, what about this: only APIs marked with i.e. a StateSync annotation will actually use this feature (APIs like ClientExports.hide, display, etc). Otherwise, we are passing back and forth serialized ClientState and ServerState instances with each and every network message between the client and the server, regardless if the API call was even able to change state in these objects.

I'm surprised that this would be possible. I thought that we have cases where we allow changes to gather (e.g. frame scoping notifications) and it is very hard to predict which API might hit something that depends on this state. For example, stream processing is very sensitive to frames visibility for detecting what and when to flush, but if we don't process pending frame scoping notifications this seems like it might break.

#85 - 03/04/2016 04:24 PM - Greg Shah

> I will document them here (and in the P2J code, where the attributes are defined).

Good.

If you want to merge to trunk, you can go ahead. I don't want to keep these around too long since they can be hard to merge.

#86 - 03/04/2016 04:37 PM - Constantin Asofiei

Greg Shah wrote:

> I'm surprised that this would be possible. I thought that we have cases where we allow changes to gather (e.g. frame scoping notifications) and it is very hard to predict which API might hit something that depends on this state. For example, stream processing is very sensitive to frames visibility for detecting what and when to flush, but if we don't process pending frame scoping notifications this seems like it might break.

I was thinking that the number of APIs to annotate (and thus control) would be less if this reverse approach was used. But on a second thought, each API which will be annotated should be reviewed carefully - if for example it can calling a pause or error messages can be shown, then it can't be discarded.

But the other idea - to discard the StateSync part from the message completely (thus don't even call getChanges or applyChanges) should be OK, if the API implementation is simple and doesn't involve any other UI part.

#87 - 03/04/2016 04:38 PM - Constantin Asofiei

Greg Shah wrote:

> I will document them here (and in the P2J code, where the attributes are defined).

Good.

If you want to merge to trunk, you can go ahead. I don't want to keep these around too long since they can be hard to merge.

I'll do it tomorrow, after I add the documentation, too.

#88 - 03/04/2016 04:44 PM - Greg Shah

> But the other idea - to discard the StateSync part from the message completely (thus don't even call getChanges or applyChanges) should be OK, if the API implementation is simple and doesn't involve any other UI part.

I suspect that there are more APIs that could depend on it than those where it can be safely avoided. If that is correct, then marking the ones that are ignoring state sync would be safer, because then the "bypass" marker and the method that is known to be safe are kept in the same location. In other words, if a future editor makes changes to that API (that make it unsafe), they will see that it is marked as a bypass and hopefully they will know to remove the marker.

On the other hand, if most APIs are safe for the bypass, then I wouldn't want to edit all of them to add the marker. You'll have to make the determination.

One other thought: the use of annotations will add runtime overhead to check for the annotation (using reflection). Are you sure that the overhead of that reflection is less than the average serialization overhead for pending changes?

#89 - 03/04/2016 04:52 PM - Constantin Asofiei

Greg Shah wrote:

> On the other hand, if most APIs are safe for the bypass, then I wouldn't want to edit all of them to add the marker. You'll have to make the determination.

OK, I understand.

> One other thought: the use of annotations will add runtime overhead to check for the annotation (using reflection). Are you sure that the overhead of that reflection is less than the average serialization overhead for pending changes?

What I'm planning to do is enhance the RoutingKey with a flag which will be set only if the annotation is present, at the time the RoutingKey is built. So, the runtime will have just a flag check - the overall impact will be minimal, no reflection involved.

Also, I don't plan to set any flags at the annotation, the simple presence of the annotation is enough.

#90 - 03/04/2016 04:56 PM - Greg Shah

> So, the runtime will have just a flag check - the overall impact will be minimal, no reflection involved.

Good idea.

#91 - 03/05/2016 07:47 AM - Constantin Asofiei

rules for attributes:

1. all size-related attributes setters change the AUTO-RESIZE to FALSE (regardless if is a no-op). With an exception for COMBO-BOX: the HEIGHT attributes do not do this... not sure why.
2. SCREEN-VALUE:
   - for SLIDER widget, getter sets the MODIFIED attribute to TRUE
   - EDITOR/SELECTION-LIST/TOGGLE-BOX: setter changes the MODIFIED to TRUE
   - EDITOR: if current is ? and setter changes it to ?, then it becomes ""
3. LABEL/FONT: setters for these may or may not affect the widget size (via auto-resize I think)
4. VISIBLE/HIDDEN: no-op can be only if the widget is already VISIBLE or HIDDEN (i.e. is realized)
5. SLIDER:NO-CURRENT-VALUE: setter call with "yes" (same as before) sets it to no? unrealized widget
6. SLIDER:TIC-MARKS - resizes the widget
7. INNER-LINES - even if the lines are the same, this can't be a no-op. It might be related to realizing the widget.
   - COMBO-BOX: if the initial assign value is zero, it defaults to 3
   - EDITOR: if the initial assign value is zero, it defaults to 4
   - SELETION-LIST: it sets the AUTO-RESIZE to FALSE
8. EDITOR: is more complex, one or more attributes may realize it by just accessing it.

#92 - 03/05/2016 07:47 AM - Constantin Asofiei

2672a rev 10981 contains comments and some other changes (mostly rollbacks).

#93 - 03/07/2016 10:12 AM - Greg Shah

Code Review Task Branch 2672a Revision 10981

The changes are fine.

How much more work is needed on this? We need you to move on to other word, unless the remaining changes are impactful.

#94 - 03/07/2016 10:32 AM - Constantin Asofiei

Greg Shah wrote:

> How much more work is needed on this? We need you to move on to other word, unless the remaining changes are impactful.

2672a/10981 can be released. Next work (related to networked exported APIs/StateSync) can be released into 1811t.

#95 - 03/07/2016 10:43 AM - Greg Shah

Please rebase to trunk 10979 and then rerun regression testing. Assuming it passes, you can merge to trunk.

#96 - 03/07/2016 10:44 AM - Greg Shah

> Next work (related to networked exported APIs/StateSync) can be released into 1811t.

Is this work higher priority than the editing improvements?

#97 - 03/07/2016 10:46 AM - Constantin Asofiei

Greg Shah wrote:

> > Next work (related to networked exported APIs/StateSync) can be released into 1811t.
>
> Is this work higher priority than the editing improvements?

Yes, it's what's left for the improvements. I'll finish it in the next 2 days. For now, I'm using an approach like this, to cut time: for test 454, identify which APIs can be marked as "internal" and don't use StateSync for them.

#98 - 03/07/2016 11:01 AM - Constantin Asofiei

Greg Shah wrote:

> Please rebase to trunk 10979 and then rerun regression testing.

Rebased and runtime testing started.

#99 - 03/08/2016 05:16 AM - Constantin Asofiei

Testing passed and 2672a rev 10982 was merged to trunk rev 10980. Branch 2672a was archived.

#100 - 03/08/2016 05:32 AM - Constantin Asofiei

Greg, I can continue with the state-sync changes, but (again) this doesn't solve fully our scenario. The most used APIs after fully initializing test 454

window (i.e. click through every tab) are these (without the 2672a branch changes,tested in 1811t):

```
10 public abstract void com.goldencode.p2j.ui.ClientExports.pauseBeforeHide(int,java.lang.String,int)
14 public abstract void com.goldencode.p2j.ui.ClientExports.refreshScrollRow(int,boolean,boolean,boolean,boole
an,int)
14 public abstract int com.goldencode.p2j.ui.ClientExports.getFocus()
16 public abstract com.goldencode.p2j.util.logical com.goldencode.p2j.ui.ClientExports.moveAfterTabItem(int,in
t,int)
20 public abstract void com.goldencode.p2j.ui.ClientExports.queryRepositioned(int,boolean,int)
24 public abstract void com.goldencode.p2j.ui.ClientExports.pushMenuDescription(com.goldencode.p2j.ui.MenuDesc
ription[])
24 public abstract boolean com.goldencode.p2j.ui.ClientExports.setWaitState(com.goldencode.p2j.ui.MousePointer
)
24 public abstract com.goldencode.p2j.ui.ScreenBuffer com.goldencode.p2j.ui.ClientExports.enable(int,int[],com
.goldencode.p2j.ui.ScreenBuffer,boolean,int)
25 public abstract boolean com.goldencode.p2j.ui.ClientExports.isInBatchMode()
31 public abstract com.goldencode.p2j.ui.ScreenBuffer com.goldencode.p2j.ui.ClientExports.apply(int,int,com.go
ldencode.p2j.ui.ScreenBuffer,com.goldencode.p2j.ui.EventList,boolean)
32 public abstract java.lang.String com.goldencode.p2j.ui.ClientExports.getFontKey(int)
32 public abstract int com.goldencode.p2j.ui.ClientExports.getTextWidthPixels(java.lang.String,int)
43 public abstract java.lang.Boolean com.goldencode.p2j.ui.ClientExports.moveToTop(int)
48 public static com.goldencode.p2j.net.RoutingKey com.goldencode.p2j.net.Dispatcher.getRoutingKey(java.lang.S
tring,java.lang.String)
48 public abstract void com.goldencode.p2j.ui.ClientExports.pushWindow(com.goldencode.p2j.ui.WindowConfig)
65 public abstract boolean com.goldencode.p2j.ui.ClientExports.hide(int,boolean,boolean)
91 public abstract com.goldencode.p2j.ui.ScreenBuffer com.goldencode.p2j.ui.ClientExports.view(int,com.goldenc
ode.p2j.ui.ScreenBuffer,int[],boolean,int)
191 public abstract com.goldencode.p2j.ui.ScreenBuffer com.goldencode.p2j.ui.ClientExports.enable(int,int,com.
goldencode.p2j.ui.ScreenBuffer,boolean,int)
455 public abstract java.lang.String com.goldencode.p2j.util.LowLevelStream.readLn(int) throws java.io.EOFExce
ption,java.io.IOException,java.lang.InterruptedException
959 public abstract com.goldencode.p2j.ui.WidgetConfig[] com.goldencode.p2j.ui.ClientExports.pushScreenDefinit
ion(com.goldencode.p2j.ui.ScreenDefinition[])
2 public abstract char com.goldencode.p2j.directory.Directory.getChar(int,java.lang.String,char)
3 public abstract int com.goldencode.p2j.directory.Directory.getInt(int,java.lang.String,int)
4 public abstract java.lang.String com.goldencode.p2j.directory.Directory.getString(int,java.lang.String,java.
lang.String)
29 public abstract com.goldencode.p2j.ui.TriggerReturnValue com.goldencode.p2j.ui.ServerExports.trigger(int,in
t,int,int,long,com.goldencode.p2j.ui.ScreenBuffer)
33 public abstract int com.goldencode.p2j.ui.ServerExports.getCurrentRow(int)
34 public abstract int com.goldencode.p2j.ui.ServerExports.setCurrentRow(int,int)
67 public abstract boolean com.goldencode.p2j.ui.ServerExports.validResource(long)
78 public abstract com.goldencode.p2j.util.BaseDataType[][] com.goldencode.p2j.ui.ServerExports.getRows(int,in
t,int,boolean)
207 public abstract int[] com.goldencode.p2j.ui.ServerExports.getLegacyTextMetrics(java.lang.String,java.lang.
String)
208 public abstract com.goldencode.p2j.ui.ScreenBuffer com.goldencode.p2j.ui.ServerExports.refresh(int)
```

The total count of "enable/disable OS events" calls reaching the JS side is ~75% of all JS API calls (i.e. ~4.5k from ~6k). Another approach would be to bracket the TC.waitForEvent calls in enable/disable OS events calls, like in this diff, but this will end up invoking it on every mouse move (as JS side notifies the client side for every mouse move currently):

```
### Eclipse Workspace Patch 1.0
#P p2j
Index: src/com/goldencode/p2j/ui/chui/ThinClient.java
===================================================================
--- src/com/goldencode/p2j/ui/chui/ThinClient.java    (revision 749)
+++ src/com/goldencode/p2j/ui/chui/ThinClient.java    (working copy)
@@ -7150,7 +7150,21 @@
                state = ((GuiDriver) tk.getInstanceDriver()).saveDrawingState();
             }

-            evt = typeAhead.getKeystroke(1000, false);
+            // control is received back by the client-side, enable events in all windows
+            tk.enableOSEvents(WindowManager.windowIds());
+
+            try
+            {
+               evt = typeAhead.getKeystroke(1000, false);
+            }
```

```
+             finally
+             {
+                // TODO: Disabling OS events here disables all user input on the client (while
+                //       the control is on the server). This is in contrast with the original 4GL/Windows
+                //       behavior where some input is processed. For example client window can be moved
+                //       while the 4GL app doesn't process any user input instructions.
+                tk.disableOSEvents(WindowManager.windowIds());
+             }

              if (!isChui)
              {
@@ -12309,12 +12323,6 @@

       TitledWindow<?> active = WindowManager.getActiveWindow();
       cs.activeWindowId = active == null ? WidgetId._DEFAULT_WINDOW_ID : active.getId().asInt();
-
-      // TODO: Disabling OS events here disables all user input on the client (while
-      //       the control is on the server). This is in contrast with the original 4GL/Windows
-      //       behavior where some input is processed. For example client window can be moved
-      //       while the 4GL app doesn't process any user input instructions.
-      tk.disableOSEvents(WindowManager.windowIds());

       return cs;
    }
@@ -12432,9 +12440,6 @@
              }
          }

-         // control is received back by the client-side, enable events in all windows
-         tk.enableOSEvents(WindowManager.windowIds());
-
          // no config updates are incoming from the server-side: all changes are sent via
          // pushScreenDefinition API calls
       }
@@ -13071,7 +13076,22 @@
             state = ((GuiDriver) tk.getInstanceDriver()).saveDrawingState();
          }

-         evt = typeAhead.getKeystroke(seconds * 1000, honorServerEvent);
+         // control is received back by the client-side, enable events in all windows
+         tk.enableOSEvents(WindowManager.windowIds());
+
+         try
+         {
+            evt = typeAhead.getKeystroke(seconds * 1000, honorServerEvent);
+         }
+         finally
+         {
+
+            // TODO: Disabling OS events here disables all user input on the client (while
+            //       the control is on the server). This is in contrast with the original 4GL/Windows
+            //       behavior where some input is processed. For example client window can be moved
+            //       while the 4GL app doesn't process any user input instructions.
+            tk.disableOSEvents(WindowManager.windowIds());
+         }

          if (!isChui)
          {
```

Please let me know how you want me to continue. What we've discussed about avoiding the state-sync payload for certain APIs can be used as an improvement, but for test 454 case, I think it will impact only ~20% of the calls.

#101 - 03/08/2016 08:04 AM - Greg Shah

Please put primary focus on the editing improvements. We know there is much to be gained there.

While you are working on the editing improvements, we can discuss and think about alternate approaches. One thing I'm wondering: does it really help us to send the enable/disableOSEvents all the way to the JS side? If it is very costly, why not just allow the JS side to always forward events to Java and let Java drop the events during the time that events are disabled?

If there are some cases (but not all) where it does make sense to ignore events all the way to the JS side, then we can implement that way.

#102 - 03/09/2016 03:50 AM - Constantin Asofiei

I've committed the changes I've made last week for the state-sync optimization to branch 2672b (from trunk rev 10980), to not lose them. They are independent of 1811t.

#103 - 03/22/2016 11:37 AM - Greg Shah

We need to close down the web performance work, at least for now. If I understand correctly, the editing improvements are not yet ready.

Can you please provide some details:

1. How much time is needed to finish the editing improvements?
2. With the editing improvements included in 1811t, how reasonable is the performance as compared to the Swing client?
3. What other high priority areas of investigation are left to explore?

#104 - 03/23/2016 04:19 PM - Greg Shah

- % Done changed from 0 to 40

- Target version changed from Milestone 12 to Milestone 17

- Parent task deleted (#1811)

#105 - 03/23/2016 05:40 PM - Constantin Asofiei

1811t contains a stable version for EDITOR which fixes PASTE and content parsing, plus some other misc fixes related to editor and paste, plus the deadlock related to window iconify. What remains:

1. an existing regression where EDITOR:FG/BGCOLOR is not honored
2. when a full line is selected, the NL char is included, too... this should be included only when multi-lines (separate with actual NL char) selection is used. This can be seen in Swing client, too, and is an existing issue.
3. optimization for key input (i.e. line spliting, what Editor.splitLine does).

What else I tried:

1. drawing optimization at the P2J client side, to i.e. draw only the affected lines (and not the entire editor). For small editors (with small viewport), drawing is already optimized so only the lines which visible are being drawn. The target were the large editors. The showstopper was when I noticed that PaintEvents are "consumed" if raised in EditorGuiImpl.setEnabled - I needed this to i.e. redraw the entire widget, if its state got changed.
2. I was inclined to keep a JS-side state of the editor (think cursor position, text), and re-measure the text if a key is pressed (and send this as an additional payload, together with the key), but there is not much gain at the client-side.

Sergey: is there a limit to how much data the Web client can hold in the CLIPBOARD? Not sure what the threshold is, but at ~10k chars (more or less) the text can't be pasted into the editor, even if its copied from outside the browser.

1. How much time is needed to finish the editing improvements?

4-5 days I think for the key input optimization.

> 2. With the editing improvements included in 1811t, how reasonable is the performance as compared to the Swing client?

PASTE is definitely a lot faster.

> 3. What other high priority areas of investigation are left to explore?

Beside what I noted above, I don't see anything else at this time.

#106 - 03/23/2016 07:29 PM - Greg Shah

> 4-5 days I think for the key input optimization.

Is it worth doing this now or should we postpone it? We have some urgent work to complete to try to get all 'new development" finished (M12 done) and the performance work is not the highest priority right now.

If we can get a significant performance improvement with these pending changes, then perhaps it is worth finishing them now. In other words, if it makes the client much more usable then we may want to finish this now.

#107 - 03/24/2016 01:49 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> Sergey: is there a limit to how much data the Web client can hold in the CLIPBOARD? Not sure what the threshold is, but at ~10k chars (more or less) the text can't be pasted into the editor, even if its copied from outside the browser.

I didn't implement the threshold, if the clipboard text size is more than maxBinaryMessage constant defined by webClient/maxBinaryMessage with the default value 32kb, then the large text is split in chunks and is send by MSG_PARTIAL packet. In my test if the data is large, the receiving time is measured in hours: for each character inserted there is a request for text width and height to the JS client from the Java side.

#108 - 03/24/2016 11:30 AM - Constantin Asofiei

1811t rev 11079 contains second pass at EDITOR improvements - a good way to improve this is to have intermediate line widths, resolved by the P2J client-side when needed, and cached; also, when possible, these are resolved in batches, not one-by-one, to reduce the JS trips.

What is left (from my previous note):

- when a full line is selected, the NL char is included, too... this should be included only when multi-lines (separate with actual NL char) selection is used. This can be seen in Swing client, too, and is an existing issue, and it affects how PASTE works.
- an existing regression where EDITOR:FG/BGCOLOR is not honored
- optimize EditorGuiImpl.mergeLines and Editor.splitLine - but this can be done on a later pass.

Sergey: please do some tests with PASTE and see if you find any issues.

Greg Shah wrote:

> 4-5 days I think for the key input optimization.

> Is it worth doing this now or should we postpone it? We have some urgent work to complete to try to get all 'new development" finished (M12 done) and the performance work is not the highest priority right now.

What is left can be done in another pass (and I'm also a little out of ideas how to optimize it, as these two APIs are more complex).

> If we can get a significant performance improvement with these pending changes, then perhaps it is worth finishing them now. In other words, if it makes the client much more usable then we may want to finish this now.

What worries me is the PASTE bug I mentioned above. But we can have a separate task for it.

#109 - 03/24/2016 01:19 PM - Sergey Ivanovskiy

- File 11080.txt added

Constantin, the partial message code didn't work correctly due to this silly mistake. Fixed and tested it with maxBinaryMessage=16 (it should be more 10). It seems that it works now. Committed revision 11080. Please look it. Your commit improves the web editor and it works tremendously faster now.

#110 - 03/25/2016 07:53 AM - Constantin Asofiei

Hynek, a heads-up about ScrollPaneGuiImpl.draw. In 1811t, I've moved this code:

```
      ThinClient.getInstance().eventBracket(true, new Runnable()
      {
         @Override
         public void run()
         {
            // resize viewport and adjust scroll bars
            adjustScrollLayoutImpl();
         }
      });
```

into a separate API (ScrollPaneGuiImpl.adjustScrollLayout()). This was needed because, if the editor's scrollbars are set dynamically, via attributes (before widget has been realized), then the ScrollPane doesn't re-compute its width until it was drawn - which resulted in incorrect dimension reported, when splitting the editor's content into lines (as word-wrap was needed). Currently, I've called this explicitly from EditorGuiImpl.processScrollPane, but we might need a more generic approach.

Also, ScrollPaneGuiImpl.createScrollBars, may be called multiple times, with different bars parameter, until the widget has been realized - and, if a scrollbar is removed, then the other one (if already created) has to be destroyed.

Please take a look at this two changes and let me know if you see anything wrong.

#111 - 03/25/2016 09:48 AM - Constantin Asofiei

1811t rev 11082 fixes the EDITOR:BGCOLOR/FGCOLOR problem in note 108.

#112 - 03/25/2016 10:17 AM - Greg Shah

> What is left can be done in another pass (and I'm also a little out of ideas how to optimize it, as these two APIs are more complex).

OK. Please document things that you tried which did not work out. Also if you have any prototype code that was useful, but not ready for check-in, we would want to keep that somewhere (maybe attached to this task as a zip).

> What worries me is the PASTE bug I mentioned above. But we can have a separate task for it.

Please create a sub-task of #2677 for this.

Is there anything else to do on this task at this time?

#113 - 03/25/2016 10:23 AM - Constantin Asofiei

Greg Shah wrote:

> What worries me is the PASTE bug I mentioned above. But we can have a separate task for it.

> Please create a sub-task of #2677 for this.

See #3039

> Is there anything else to do on this task at this time?

What's left is an ArrayIndexOutOfBoundsException I've seen sometimes when editing, in EditorGuiImpl.getTextWidthNative:2888, I'm trying to fix this.

#114 - 03/25/2016 10:30 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, a heads-up about ScrollPaneGuiImpl.draw. In 1811t, I've moved this code:
> [...]
> into a separate API (ScrollPaneGuiImpl.adjustScrollLayout()). This was needed because, if the editor's scrollbars are set dynamically, via attributes (before widget has been realized), then the ScrollPane doesn't re-compute its width until it was drawn - which resulted in incorrect dimension reported, when splitting the editor's content into lines (as word-wrap was needed). Currently, I've called this explicitly from EditorGuiImpl.processScrollPane, but we might need a more generic approach.
>
> Also, ScrollPaneGuiImpl.createScrollBars, may be called multiple times, with different bars parameter, until the widget has been realized - and, if a scrollbar is removed, then the other one (if already created) has to be destroyed.

Following the same logic added to SPGI.createScrollBars() shouldn't the NONE case destroy the bars if existing (and clear the both flag)?

#115 - 03/25/2016 10:31 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Also, ScrollPaneGuiImpl.createScrollBars, may be called multiple times, with different bars parameter, until the widget has been realized - and, if a scrollbar is removed, then the other one (if already created) has to be destroyed.

Following the same logic added to SPGI.createScrollBars() shouldn't the NONE case destroy the bars if existing (and clear the both flag)?

You are correct, I'll fix it.

#116 - 03/25/2016 12:06 PM - Constantin Asofiei

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > Also, ScrollPaneGuiImpl.createScrollBars, may be called multiple times, with different bars parameter, until the widget has been realized - and, if a scrollbar is removed, then the other one (if already created) has to be destroyed.
>
> Following the same logic added to SPGI.createScrollBars() shouldn't the NONE case destroy the bars if existing (and clear the both flag)?
>
> You are correct, I'll fix it.

The fix for this is in 1811t rev 11084.

#117 - 03/31/2016 04:08 PM - Constantin Asofiei

1811t rev 10992 fixes the EDITOR regressions introduced by previous changes.

#118 - 04/01/2016 09:22 AM - Greg Shah

All current work on this is done, correct?

#119 - 04/01/2016 09:24 AM - Constantin Asofiei

Greg Shah wrote:

> All current work on this is done, correct?

Correct.

#120 - 04/01/2016 09:38 AM - Greg Shah

OK, we will defer additional work on this until later.

#121 - 04/01/2016 11:19 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> 1811t rev 10992 fixes the EDITOR regressions introduced by previous changes.

On version 10991 I got this exception on the server side

```
Caused by: java.lang.NullPointerException
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver$TextMetrics.getWidths(AbstractGuiDriver.j
ava:2931)
        at com.goldencode.p2j.ui.client.gui.driver.web.GuiWebDriver.getTextWidths(GuiWebDriver.java:1686)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl.computeLineWidths(EditorGuiImpl.java:2832)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl.maxLineWidthNative(EditorGuiImpl.java:2628)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl.access$23(EditorGuiImpl.java:2623)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl$EditorScrollContainer.getScrollDimension(EditorGuiIm
pl.java:3406)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.adjustScrollLayoutImpl(ScrollPaneGuiImpl.java:65
9)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.access$0(ScrollPaneGuiImpl.java:652)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl$1.run(ScrollPaneGuiImpl.java:163)
        at com.goldencode.p2j.ui.chui.ThinClient.eventBracket(ThinClient.java:14119)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.adjustScrollLayout(ScrollPaneGuiImpl.java:157)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.draw(ScrollPaneGuiImpl.java:183)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl$2.run(EditorGuiImpl.java:776)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.EditorGuiImpl.draw(EditorGuiImpl.java:766)
        at com.goldencode.p2j.ui.client.Editor.draw(Editor.java:792)
        at com.goldencode.p2j.ui.client.widget.AbstractContainer.draw(AbstractContainer.java:414)
        at com.goldencode.p2j.ui.client.widget.Viewport.draw(Viewport.java:64)
        at com.goldencode.p2j.ui.client.widget.AbstractContainer.draw(AbstractContainer.java:414)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.access$4(BorderedPanelGuiImpl.java:1)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1$1.run(BorderedPanelGuiImpl.java:200)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1.run(BorderedPanelGuiImpl.java:189)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.draw(BorderedPanelGuiImpl.java:136)
```

```
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl$2.run(ScrollPaneGuiImpl.java:200)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.draw(ScrollPaneGuiImpl.java:191)
        at com.goldencode.p2j.ui.client.gui.FrameGuiImpl$2.run(FrameGuiImpl.java:478)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.FrameGuiImpl.draw(FrameGuiImpl.java:464)
        at com.goldencode.p2j.ui.client.Frame.draw(Frame.java:1992)
        at com.goldencode.p2j.ui.client.widget.AbstractContainer.draw(AbstractContainer.java:414)
        at com.goldencode.p2j.ui.client.widget.Viewport.draw(Viewport.java:64)
        at com.goldencode.p2j.ui.client.widget.AbstractContainer.draw(AbstractContainer.java:414)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.access$4(BorderedPanelGuiImpl.java:1)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1$1.run(BorderedPanelGuiImpl.java:200)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1.run(BorderedPanelGuiImpl.java:189)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.draw(BorderedPanelGuiImpl.java:136)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl$2.run(ScrollPaneGuiImpl.java:200)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.ScrollPaneGuiImpl.draw(ScrollPaneGuiImpl.java:191)
        at com.goldencode.p2j.ui.client.gui.WindowWorkSpace.draw(WindowWorkSpace.java:87)
        at com.goldencode.p2j.ui.client.widget.AbstractContainer.draw(AbstractContainer.java:414)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.access$4(BorderedPanelGuiImpl.java:1)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1$1.run(BorderedPanelGuiImpl.java:200)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl$1.run(BorderedPanelGuiImpl.java:189)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2161)
        at com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.draw(AbstractGuiDriver.java:2038)
        at com.goldencode.p2j.ui.client.gui.BorderedPanelGuiImpl.draw(BorderedPanelGuiImpl.java:136)
        at com.goldencode.p2j.ui.client.Window.draw(Window.java:1374)
        at com.goldencode.p2j.ui.client.gui.WindowGuiImpl.draw(WindowGuiImpl.java:560)
        at com.goldencode.p2j.ui.client.OutputManager.setInvalidate(OutputManager.java:1275)
        at com.goldencode.p2j.ui.chui.ThinClient.eventDrawingBracket(ThinClient.java:14067)
        at com.goldencode.p2j.ui.chui.ThinClient.independentEventDrawingBracket(ThinClient.java:13933)
        at com.goldencode.p2j.ui.client.widget.AbstractWidget.processSystemKey(AbstractWidget.java:1990)
        at com.goldencode.p2j.ui.chui.ThinClient.checkForSystemEvent(ThinClient.java:13299)
        at com.goldencode.p2j.ui.chui.ThinClient.waitForEvent(ThinClient.java:13188)
        at com.goldencode.p2j.ui.chui.ThinClient.waitForWorker(ThinClient.java:11059)
        at com.goldencode.p2j.ui.chui.ThinClient.waitFor(ThinClient.java:10633)
        at com.goldencode.p2j.ui.chui.ThinClient.waitFor(ThinClient.java:10587)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:76)
        at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:705)
        at com.goldencode.p2j.net.Conversation.block(Conversation.java:319)
        at com.goldencode.p2j.net.Conversation.waitMessage(Conversation.java:257)
        at com.goldencode.p2j.net.Queue.transactImpl(Queue.java:1132)
        at com.goldencode.p2j.net.Queue.transact(Queue.java:589)
        at com.goldencode.p2j.net.BaseSession.transact(BaseSession.java:223)
        at com.goldencode.p2j.net.HighLevelObject.transact(HighLevelObject.java:163)
        at com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(RemoteObject.java:1425)
        at com.goldencode.p2j.net.InvocationStub.invoke(InvocationStub.java:97)
        at com.sun.proxy.$Proxy11.standardEntry(Unknown Source)
        at com.goldencode.p2j.main.ClientCore.start(ClientCore.java:281)
        at com.goldencode.p2j.main.ClientCore.start(ClientCore.java:102)
        at com.goldencode.p2j.main.ClientDriver.start(ClientDriver.java:202)
        at com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:396)
        at com.goldencode.p2j.main.ClientDriver.process(ClientDriver.java:1)
        at com.goldencode.p2j.main.ClientDriver.main(ClientDriver.java:265)
```

But I can't reproduce it for the current 10992 version. Correct?

#122 - 04/01/2016 11:20 AM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> Constantin Asofiei wrote:
>
> > 1811t rev 10992 fixes the EDITOR regressions introduced by previous changes.
>
> On version 10991 I got this exception on the server side
> [...]
> But I can't reproduce it for the current 10992 version. Correct?

Correct, is fixed in 10992.

#123 - 11/16/2016 12:30 PM - Greg Shah

- Target version changed from Milestone 17 to Performance and Scalability Improvements

#124 - 01/03/2018 02:05 PM - Greg Shah

- Related to Feature #3246: reduce the amount of data being sent to the client-side when an UI attribute is being changed added

Files

restore.png	46.7 KB	01/31/2016	Sergey Ivanovskiy
performance_issue_1.txt	2.85 KB	02/12/2016	Sergey Ivanovskiy
performance_issue_2.txt	5.31 KB	02/13/2016	Sergey Ivanovskiy
performance_issue_3.txt	28.7 KB	02/15/2016	Sergey Ivanovskiy
icons_bg_issue.png	21.3 KB	02/16/2016	Sergey Ivanovskiy
browse.png	24.7 KB	02/16/2016	Sergey Ivanovskiy
browse_artifacts_in_title.png	69.1 KB	02/16/2016	Sergey Ivanovskiy
LineStrokeTest2.html	6 KB	02/17/2016	Sergey Ivanovskiy
performance_issue_4.txt	29.3 KB	02/17/2016	Sergey Ivanovskiy
jsclip.html	846 Bytes	02/19/2016	Constantin Asofiei
js-clip.png	2.21 KB	02/19/2016	Constantin Asofiei
11080.txt	771 Bytes	03/24/2016	Sergey Ivanovskiy