User Interface - Feature #2967

web socket message sizes are limited to 32K, make sure that drawing ops and images larger than this are handled properly

01/25/2016 05:13 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	GUI Support for a Complex ADM2 App		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to User Interface - Feature #1811: implement the AJAX client driver	**Closed**	**11/01/2013**	**03/06/2014**

History

#1 - 01/25/2016 05:13 PM - Greg Shah

Put the fix for this into branch 1811t.

#2 - 02/05/2016 10:07 AM - Sergey Ivanovskiy

Greg, please could you write the requirements for this task? Should the maximum message size be configured via the directory under the webClient node or clientConfig? Do you suppose to split messages that more than 32K?

#3 - 02/05/2016 10:33 AM - Greg Shah

We should not have a message limit. If it goes above 32K (the inherent limit in websockets), we should send multiple messages and "put them back together" on the other side. This may have to be done in both directions to be safe, though it is the Java to JS direction is the one which will be most likely to hit this limit (e.g. sending a large image or font).

#4 - 02/05/2016 01:13 PM - Sergey Ivanovskiy

If my understanding is correct, then Firefox Web Socket API doesn't require splitting data, if it sends/reads data objects from/to the opened web socket. In contrast the Jetty Web Socket API has these methods to send data chunks

```
while(chunk is not last)
{
  remote.sendPartialBytes(chunk,false);
}
remote.sendPartialBytes(lastChunk,true);
```

Thus it seems that we should change only PushMessagesWorker.

#5 - 02/06/2016 03:20 PM - Sergey Ivanovskiy

In any case if the websocket on the java side doesn't support messages that exceed the 32K limit, then we need to split messages to parts within this limit. Thus we can implement a partial message type, that can be handled separately to gather the complete message as an analog of the Jetty Websocket API. In this task I am planning to move the JS client to use Web Workers API.

#6 - 02/07/2016 12:33 PM - Sergey Ivanovskiy

Following this plan:
1) Create the common module for the input and output operations - p2j.io.js
2) Reuse p2j.io.js API in p2j.socket.js module
3) Create the common connector module p2j.connector.js that will be started in the web worker thread and will support web socket communication with the client web server. The p2j.io.js module will be reused by p2j.connector.js
4) Change p2j.socket.js to be a layer between UI modules and p2j.connector.js, a websocket transport module. Use transferable objects in order to increase the round trip time between JS UI thread and the web worker thread. Each message between these two modules will be encoded in the message object with prepared data according to each message type:

```
{
   type: type,
   field1: value1,
.................
   fieldn: valuen
}
```

where values can be Numbers, Booleans or ArrayBuffers.

#7 - 02/08/2016 12:14 PM - Greg Shah

> Use transferable objects in order to increase the round trip time between JS UI thread and the web worker thread.

Can you explain this a bit more?

Why would we want to increase the round trip time?

> Each message between these two modules will be encoded in the message object with prepared data according to each message type

The idea is that p2j.connector.js will parse each message?

Please create helper functions as needed to use more common code for the reading/writing of messages.

I would also like to see message types as a symbolic names like we do with p2j.screen.ops for the drawing ops. This should be used in both p2j.connector.js and p2j.socket.js, I guess.

#8 - 02/08/2016 12:41 PM - Sergey Ivanovskiy

> Use transferable objects in order to increase the round trip time between JS UI thread and the web worker thread.

Sorry, I mean "decrease", transferable objects decrease the overhead of the message delivery from the UI thread to the web worker thread by webWorker.postMessage(message) and from the web worker thread to the UI thread by self.postMessage(message).

> The idea is that p2j.connector.js will parse each message?

Yes, the websocket will be created from the web worker thread and web socket messages will be handled first by the web worker thread, and then they will be delivered to the UI thread. If the message type is 0x81 (draw), the binary array will be delivered to the UI thread. Since the transferable objects don't require to create and to copy their contents, the overhead of the proposed changes should be minimal. I am planning to test these changes thoroughly.

#9 - 02/08/2016 12:45 PM - Sergey Ivanovskiy

> I would also like to see message types as a symbolic names like we do with p2j.screen.ops for the drawing ops. This should be used in both p2j.connector.js and p2j.socket.js, I guess.

Okay, planning to do it.

#10 - 02/08/2016 02:03 PM - Greg Shah

The plan sounds reasonable.

#11 - 02/09/2016 09:13 AM - Sergey Ivanovskiy

The code changes are ready. The test with Firefox is passed. Planning to add comments and to commit within two or more hours if the tests with Chrome and IE will be successful.

#12 - 02/09/2016 02:12 PM - Sergey Ivanovskiy

I encountered troubles if we use the Jetty partial message API remote.sendPartialBytes(data,isLast), then the JS web socket throws exception on

receiving the partial messages that should form the whole message. Thus it is not obvious if the browser supports the partial messages? The next question has no answer if we have 32 * 1024 b limit why we can send the messages that exceed this limit, but we already send such messages to the JS side ~ 600 * 1024b according to the log. I don't know if it is required to split messages.

#13 - 02/09/2016 02:17 PM - Sergey Ivanovskiy

Greg, should I commit my changes without remote.sendPartialBytes(data,isLast)?

#14 - 02/09/2016 02:34 PM - Sergey Ivanovskiy

For an example, ./demo/sample2.bmp has 43.9 kB (43,854 bytes) that exceeds the limit returned by session.getPolicy().getMaxBinaryMessageSize();="32K". But this image is successfully loaded by the existing code.
Now I made changes to open web socket connections in the dedicated web worker thread. Does it need to commit?

#15 - 02/10/2016 12:40 PM - Sergey Ivanovskiy

Since the Web Socket Jetty API and web socket browser's client don't work as it is expected. This article http://www.lenholgate.com/blog/2011/07/websockets-is-a-stream-not-a-message-based-protocol.html and its critics https://news.ycombinator.com/item?id=3377406 explain why the browser's web socket client probably doesn't support messages in a sequence of web socket frames. Planing to write simple Jetty server and JS client to investigate this issue more thoroughly. It should help to explain the server's and client's limits for sending/receiving long messages.

#16 - 02/10/2016 12:59 PM - Sergey Ivanovskiy

- File websocket_config.png added

My Firefox websocket configuration parameters are in this picture. network.websocket.max-message-size is related to this issue. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#Gecko_11.0 thus 2Gb is a theoretical maximum.

#17 - 02/10/2016 01:18 PM - Sergey Ivanovskiy

- File web_worker_1.txt added

Sergey Ivanovskiy wrote:

> Greg, should I commit my changes without remote.sendPartialBytes(data,isLast)?

The proposed changes are in this diff

#18 - 02/11/2016 02:05 PM - Sergey Ivanovskiy

According to my test if we send the file with its size exceeding the maximum binary message size excepted by Jetty (given by session.getPolicy().getMaxBinaryMessageSize()), then
the Jetty server throws Binary message size [1173069] exceeds maximum size [32768]. Thus we must split the large message to parts, if it exceeds the limit defined by the Embedded Jetty server policy, but it is not required if the large message is send to the JS client due to the note 16. The JS client doesn't support partial frames formed by the Jetty Web Socket API.

#19 - 02/12/2016 12:41 AM - Sergey Ivanovskiy

Found that on the current branch the draw buffer size (0x81 message) is large enough if we scroll down the editor content by the mouse or the keyboard. The reason that confuses me to follow the proposed plan is the degraded performance of the current branch. If we would decide to implement splitting messages on the JS client, then we were required to implement the web worker and to add a new partial message type. The opened questions are

1. what kind of large data do we expect to get from the client?
2. Could we send files or clipboard content?
3. Usually we get small requests from the client. Thus, we need to interleave short messages with partial messages in order the performance doesn't degrade and we can deal with application while sending the large data. Does it require a user's notification from the JS client?
4. Does it require to implement a web server side buffer for incomplete messages?

#20 - 02/15/2016 11:51 AM - Greg Shah

Some thoughts:

1. The current RFC implements a per "frame" payload size specified in the frame header with a maximum 64-bit integer. For our purposes, this is essentially an unlimited size.

https://tools.ietf.org/html/rfc6455#page-27

2. Your findings are consistent with the fact that each implementation (a different one for each browser and the server-side Jetty) have their own potential behaviors and limits.

3. If I understand things correctly, our server-side code has the maximum text and binary message sizes set as annotations:

```
@WebSocket(maxIdleTime = 900000, maxTextMessageSize = 4096, maxBinaryMessageSize = 32768)
public class WebClientProtocol
```

I think that 32K is not a good choice. I haven't checked our text message usage to know if 4096 is safe. I don't like implementing important values like this in annotations, which are hard coded. I'd rather we use the programmatic interfaces in Jetty and later we can make things configurable (e.g. a server-side max size override in the directory).

http://stackoverflow.com/questions/17497173/jetty-9-websocket-server-max-message-size-on-session

Our default value should probably be higher than 32K. But there will always be a scenario where we are trying to send something larger than our chosen limit. I think we need to implement our own partial message support for that case.

4. I don't understand all your findings so far. In particular, you state in notes 12 and 14 above, that our current server-side code sends data that is larger than 32K. But then in note 18 you find that we get an exception when trying to send something greater than 32K. I must be missing something here, because this seems to be a contradiction.

5. I also don't understand what you mean by "The reason that confuses me to follow the proposed plan is the degraded performance of the current branch.". Are you saying that your proposed diffs will degrade performance? Or are you saying that our current implementation has degraded performance that will be improved by your proposed diffs?

6. Although the client will **usually** be sending very small data, we will sometimes need to upload files and/or send large buffers from the clipboard. I suspect this means that both the server and the client must be prepared to deal with multi-part messages. But we will have to do our own version since the Jetty implementation won't work.

7. You state this: "we need to interleave short messages with partial messages in order the performance doesn't degrade and we can deal with application while sending the large data". Can you be more specific about which direction you are talking about here? Is this Java to JS OR JS to Java OR both?

8. In regard to the user notification in the JS client, I don't know the answer. Any sufficiently long period of unresponsiveness should provide some feedback to the user, but we would have to detect that there is going to be unresponsiveness, how do we do that? We could implement some heuristic (e.g. a rule that if we are receiving a multi-part message of greater than a certain size, then we shift to a busy cursor), but this seems arbitrary and fragile.

#21 - 02/15/2016 12:36 PM - Sergey Ivanovskiy

- File TestEmbeddedJetty.zip added

> 4. I don't understand all your findings so far. In particular, you state in notes 12 and 14 above, that our current server-side code sends data that is larger than 32K. But then in note 18 you find that we get an exception when trying to send something greater than 32K. I must be missing something here, because this seems to be a contradiction.

In the notes 12 and 14 I mean that if we try to use the Jetty API (sendPartialBytes(chunk,isLast)) to form frames with incomplete flag then the JS part fails to parse these frames. And we don't need to split messages from Java to JS since we can use a complete message by sendBytes(message) with the size bounded by 2Gb.
The note 18 states that if we send a large message from JS to Java using JS WebSocket API, then the Jetty server can fails depending on the maxBinaryMessageSize parameter.
The control rules are not symmetric and depend on the case if we send from Java to JS or from JS to Java.

> 5. I also don't understand what you mean by "The reason that confuses me to follow the proposed plan is the degraded performance of the current branch.". Are you saying that your proposed diffs will degrade performance? Or are you saying that our current implementation has degraded performance that will be improved by your proposed diffs?

No, using JS Web Worker API adds an overhead since we need to post messages from the web worker thread to the main JS UI thread and from the main JS UI thread to the web worker thread. 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.

> 6. Although the client will **usually** be sending very small data, we will sometimes need to upload files and/or send large buffers from the clipboard. I suspect this means that both the server and the client must be prepared to deal with multi-part messages. But we will have to do our own version since the Jetty implementation won't work.
>
> 7. You state this: "we need to interleave short messages with partial messages in order the performance doesn't degrade and we can deal with application while sending the large data". Can you be more specific about which direction you are talking about here? Is this Java to JS OR JS to Java OR both?

I mean the direction from JS to Java in the theoretical case if we send a large file from JS to Java. Please could you specify the use cases in which large messages should be sent from JS to Java?
I used this simple test to detect different cases but it isn't documented and intended to use as a working test case. This page http://localhost:8080 loads index.html with the client script with options to send files and to get some large binaries from the server to the client.

#22 - 02/15/2016 05:37 PM - Greg Shah

> The note 18 states that if we send a large message from JS to Java using JS WebSocket API, then the Jetty server can fails depending on the maxBinaryMessageSize parameter.
> The control rules are not symmetric and depend on the case if we send from Java to JS or from JS to Java.

OK, now I understand.

> I mean the direction from JS to Java in the theoretical case if we send a large file from JS to Java. Please could you specify the use cases in which large messages should be sent from JS to Java?

One example for sure is the clipboard. If there is large data in it (e.g. an image), then this will definitely need to be sent to the Java side from JS.

Another example is the SYSTEM-DIALOG-GET-FILE statement, where the end user would be selecting a file on the local system for upload. We are planning to work this in #1830, although:

1. We are trying to avoid this work for now. I'm not sure if we will be able to avoid it, it is the customer's choice and we are waiting to hear back.

2. It is not clear to me if the SYSTEM-DIALOG-GET-FILE statement filesystem being read is the JS side or the Java side. The way we have implemented our system, it would be consistent to be at the Java side. But I could also see customers needing a real file on the end-user's system to be uploaded from the browser.

I am not sure if we will need this large file upload from the browser or not.

> If we would decide to implement splitting messages on the JS client, then we were required to implement the web worker and to add a new partial message type.

Why must we use web workers in this case?

#23 - 02/16/2016 03:29 PM - Sergey Ivanovskiy

Agree, it doesn't require to use Web Workers API. There is an alternative to use Timers API or http://dbaron.org/log/20100309-faster-timeouts in order to send partial messages.
Greg, please review this plan:
1) Add new MSG_PARTIAL to WebClientMessageTypes having this bytes format:
[MSG_PARTIAL_TYPE][MSG_ID][PARTIAL_STATUS][PAYLOAD_TYPE][PAYLOAD_LENGTH][PAYLOAD_DATA]
where
MSG_PARTIAL_TYPE is one byte that equals to MSG_PARTIAL
MSG_ID is 4 bytes unsigned integer, a unique message id generated by browsers
PARTIAL_STATUS is 1 byte that equals 0 if it is the last message with MSG_ID or 1 if a next message with the same MSG_ID should follow this one.
PAYLOAD_TYPE is 1 byte is one of WebClientMessageTypes types (with new file content type)
PAYLOAD_LENGTH is 4 byte unsigned integer that determines the payload data length in bytes.
PAYLOAD_DATA is a binary array with the length given by PAYLOAD_LENGTH
2) The Java side should provide the JS client with the maximum binary message size supported by the Java side as a field of the global p2j object - p2j.maxBinaryMessageSize.parameter.
3) The JS client should wrap the large clipboard content or file content in MSG_PARTIAL-type messages, sending them using http://dbaron.org/log/20100309-faster-timeouts
4) The Java side should opens a file stream for a temporary file for each partial message with new MSG_ID and then appends a new payload content to this opened file stream.
5) On receiving the last message with the given id closes the file stream and delivers the whole message of clipboard or file content to the corresponding application handler
GuiWebSocket that should handle the file and clipboard content messages.

#24 - 02/19/2016 11:47 AM - Greg Shah

I'm OK with this proposed approach. Make sure you write your own from-scratch version of the faster timeouts approach.

#25 - 02/20/2016 01:18 PM - Sergey Ivanovskiy

- File 2967_1.txt added

Greg, please review the committed revision 11027.

#26 - 02/22/2016 04:52 AM - Sergey Ivanovskiy

Committed revision 11029 takes into account that a file size can be larger than the JVM heap size and fixes the close channel issue if IO exceptions occur. Tested with MSG_PASTE and MSG_FILE_UPLOAD (the code is not committed due to it is unclear what to do with uploaded files)

#27 - 02/22/2016 09:40 AM - Greg Shah

I'm reviewing the code now. However, I don't want to support the file upload at this time. As mentioned in note 22, we aren't sure that it will be needed or is correct. But we do know that large clipboard data must be supported. Please remove the file upload changes. I'm worried they are more of a security problem at this point.

#28 - 02/22/2016 10:11 AM - Greg Shah

Code Review Task Branch 1811t Revision 11029

1. I prefer not to create the temp dir for every client. This support will only be used for limited testcases and I want to use the temp dir approach on a lazy basis (only if and when needed).

2. Please move the core of the MSG_PARTIAL support in WebClientProtocol.processBinaryMessage() into a separate method. Putting it inline in WebClientProtocol.processBinaryMessage() makes the method harder to read. Better yet, a separate helper class would be a nice way to isolate this code.

3. Please don't add blank lines to the end of classes (see GenericWebServer and WebSocketConfig). Per our coding standards, we don't have an extra blank line at the beginning or end of a class.

#29 - 02/22/2016 01:27 PM - Sergey Ivanovskiy

Greg, please approve that we only need to support MSG_PASTE. In this case we aren't required to save partial messages and can process them in the same thread and the JS client must send it in one transaction in order the current focused widget isn't changed. At first I consider the asynchronous file upload as the main usecase, but it doesn't need and all design can be simplified.

#30 - 02/22/2016 01:29 PM - Sergey Ivanovskiy

I found the positive usage of the fast timer. It looks that the JS client reacts more rapidly if the setTimeout is replaced by the fast timer.

#31 - 02/22/2016 01:52 PM - Greg Shah

> please approve that we only need to support MSG_PASTE. In this case we aren't required to save partial messages and can process them in the same thread and the JS client must send it in one transaction in order the current focused widget isn't changed.

Yes, approved.

> At first I consider the asynchronous file upload as the main usecase, but it doesn't need and all design can be simplified.

Great! Please save the diffs for file upload support into this task, so that we can use it later if needed.

#32 - 02/22/2016 03:42 PM - Sergey Ivanovskiy

I encountered the thread lock in the case of the large paste data in the editor, the main thread and the web task thread are locked. The scenario is the same as it was in
#2956 change-37191 (LE: GES what does change-37191 reference?). After the paste data has been delivered to the Java side, then the editor continues to send text height and width requests to the JS client and simultaneously we minimize and restore the window. Planning to investigate and to fix it first.

#33 - 02/23/2016 01:53 AM - Sergey Ivanovskiy

#33 has been deleted due to the fact MSG_PASTE is handled by GuiWebSocket like it has been described.

LE: GES put this back in because it is a nice summary of how things work.

I think MSG_PASTE that is handled by WebClientProtocol should be processed via the clipboard implementation.
First, fill the clipboard buffer on the Web Java client, and then send Keyboard.KA_PASTE key.
We have a chain of nodes: the JS client, the Web Java client and the P2J server.
Between the JS client and the Web Java client we send MSG_PASTE with the data, that can be wrapped by MSG_PARTIAL.
But between the Web Java client and the P2J server we should send Keyboard.KA_PASTE key and then EditorGuiImpl should catch Keyboard.KA_PASTE and invokes gd.clipboardContents(). Is it correct?

#34 - 02/23/2016 08:01 AM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> I encountered the thread lock in the case of the large paste data in the editor, the main thread and the web task thread are locked. The scenario is the same as it was in
> #2956 change-37191. After the paste data has been delivered to the Java side, then the editor continues to send text height and width requests to the JS client and simultaneously we minimize and restore the window. Planning to investigate and to fix it first.

The main thread and the web worker thread are blocked in the case the main thread that owns the gui driver instance is waiting for the text dimensions while the web worker thread processing the minimize request tries to access the gui driver instance. Thus the web task with the text dimensions is waiting in the tasks queue.

```
Name: main
State: WAITING on java.lang.Object@587702f
Total blocked: 1,171  Total waited: 1,050

Stack trace:
java.lang.Object.wait(Native Method)
java.lang.Object.wait(Object.java:502)
com.goldencode.p2j.ui.client.driver.web.WebClientProtocol.waitForResult(WebClientProtocol.java:816)
com.goldencode.p2j.ui.client.gui.driver.web.GuiWebSocket.getTextHeight(GuiWebSocket.java:1616)
com.goldencode.p2j.ui.client.gui.driver.web.GuiWebEmulatedWindow.getTextHeight(GuiWebEmulatedWindow.java:297)
com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.resolveTextMetrics(AbstractGuiDriver.java:2715)
com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.getTextWidth(AbstractGuiDriver.java:1340)
com.goldencode.p2j.ui.client.widget.AbstractWidget.getTextWidth(AbstractWidget.java:2263)
com.goldencode.p2j.ui.client.widget.AbstractWidget.getTextWidthNative(AbstractWidget.java:1817)
com.goldencode.p2j.ui.client.gui.EditorGuiImpl.cursorRight(EditorGuiImpl.java:1718)
com.goldencode.p2j.ui.client.Editor.placeChar(Editor.java:2003)
com.goldencode.p2j.ui.client.gui.EditorGuiImpl.processKeyEvent(EditorGuiImpl.java:461)
com.goldencode.p2j.ui.client.widget.AbstractWidget$1.run(AbstractWidget.java:1973)
com.goldencode.p2j.ui.chui.ThinClient.eventBracket(ThinClient.java:13924)
com.goldencode.p2j.ui.chui.ThinClient.eventDrawingBracket(ThinClient.java:13865)
com.goldencode.p2j.ui.chui.ThinClient.independentEventDrawingBracket(ThinClient.java:13738)
com.goldencode.p2j.ui.client.widget.AbstractWidget.processSystemKey(AbstractWidget.java:1950)
com.goldencode.p2j.ui.chui.ThinClient.checkForSystemEvent(ThinClient.java:13104)
com.goldencode.p2j.ui.chui.ThinClient.waitForEvent(ThinClient.java:12993)
com.goldencode.p2j.ui.chui.ThinClient.waitForWorker(ThinClient.java:10870)
com.goldencode.p2j.ui.chui.ThinClient.waitFor(ThinClient.java:10451)
com.goldencode.p2j.ui.chui.ThinClient.waitFor(ThinClient.java:10405)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:497)
com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:76)
com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:705)
com.goldencode.p2j.net.Conversation.block(Conversation.java:319)
com.goldencode.p2j.net.Conversation.waitMessage(Conversation.java:257)
com.goldencode.p2j.net.Queue.transactImpl(Queue.java:1128)
```

```
com.goldencode.p2j.net.Queue.transact(Queue.java:585)
com.goldencode.p2j.net.BaseSession.transact(BaseSession.java:223)
com.goldencode.p2j.net.HighLevelObject.transact(HighLevelObject.java:163)
com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(RemoteObject.java:1425)
com.goldencode.p2j.net.InvocationStub.invoke(InvocationStub.java:97)
com.sun.proxy.$Proxy11.standardEntry(Unknown Source)
com.goldencode.p2j.main.ClientCore.start(ClientCore.java:281)
com.goldencode.p2j.main.ClientCore.start(ClientCore.java:102)
com.goldencode.p2j.main.ClientDriver.start(ClientDriver.java:201)
com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:398)
com.goldencode.p2j.main.ClientDriver.process(ClientDriver.java:95)
com.goldencode.p2j.main.ClientDriver.main(ClientDriver.java:267)


Name: webtaskworker
State: WAITING on com.goldencode.p2j.ui.client.gui.driver.web.GuiWebDriver@2f3baeaa
Total blocked: 996  Total waited: 997

Stack trace:
java.lang.Object.wait(Native Method)
java.lang.Object.wait(Object.java:502)
com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.selectWindow(AbstractGuiDriver.java:1700)
com.goldencode.p2j.ui.client.gui.driver.AbstractGuiDriver.iconifyWindow(AbstractGuiDriver.java:1887)
com.goldencode.p2j.ui.client.gui.driver.web.GuiWebDriver.windowIconified(GuiWebDriver.java:956)
com.goldencode.p2j.ui.client.gui.driver.web.GuiWebSocket.processBinaryMessage(GuiWebSocket.java:1309)
com.goldencode.p2j.ui.client.driver.web.WebClientProtocol$1.run(WebClientProtocol.java:321)
com.goldencode.p2j.ui.client.driver.web.WebTaskWorker.run(WebTaskWorker.java:79)
```

#35 - 02/23/2016 09:07 AM - Sergey Ivanovskiy

This thread issue can be observed in the case if the pasted text has no more than hundreds lines and WebClientProtocol.waitForResult(int msgId) has no timeout. Thus it can be observed in P2j if the code for WebClientProtocol.waitForResult(int msgId) would not implement a timeout.

#36 - 02/23/2016 10:17 AM - Sergey Ivanovskiy

The same thread issue can be reproduced for ./demo/movie-ratings-dynamic.p com.goldencode.testcases.demo.MovieRatingsDynamic.execute if we try to access available widgets in the middle of the editor's text processing. We need to block the JS client mouse listeners if the text processing is not finished or to implement a timeout for WebClientProtocol.waitForResult(int msgId). Constantin, what do you think?

#37 - 02/23/2016 12:04 PM - Sergey Ivanovskiy

> Code Review Task Branch 1811t Revision 11029
>
> 1. I prefer not to create the temp dir for every client. This support will only be used for limited testcases and I want to use the temp dir approach on a lazy basis (only if and when needed).
>
> 2. Please move the core of the MSG_PARTIAL support in WebClientProtocol.processBinaryMessage() into a separate method. Putting it inline in WebClientProtocol.processBinaryMessage() makes the method harder to read. Better yet, a separate helper class would be a nice way to isolate this code.
>
> 3. Please don't add blank lines to the end of classes (see GenericWebServer and WebSocketConfig). Per our coding standards, we don't have an extra blank line at the beginning or end of a class.

Committed revision 11031 should improve the code according to the review. These two known issues are blocking in this task
1) The blocking thread issue described in 34, 35, 36
2) The paste functionality is not practical for the text with no more than hundreds lines due to the fact that the character processing is very slow, 1 second per a several characters is observed from one text dimension request received by the JS client to the next request.

#38 - 02/23/2016 01:11 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> We need to block the JS client mouse listeners if the text processing is not finished

It might work if we disable mouse processing while messages are being processed on JS side - this way, JS will no longer be able to invoke the remote client side until a message is processed.

> or to implement a timeout for WebClientProtocol.waitForResult(int msgId). Constantin, what do you think?

Timeout will not work, as getTextWidth really needs a response.

For editor, we need to consider if pasting text into it may or may not result in invoking a trigger. Each character in the pasted text is simulated as a key press in P2J, so we can't easily optimize it if triggers are involved. We need some tests to find how 4GL reacts on PASTE: if there is a trigger on a char in the text, does it get invoked? The following simple tests shows that triggers are not involved when pasting text into the editor:

```
DEF VAR ch AS CHAR VIEW-AS EDITOR SIZE 20 BY 20.
DEF VAR i AS INT.
i = 0.
ON 'a':U ANYWHERE
DO:
   i = i + 1.
   MESSAGE i "a".
   RETURN.
END.

UPDATE ch.
```

Have some text with "a" chars in the clipboard and paste it into the editor: the trigger will not be executed. But this is only for GUI - the linux ChUI client does invoke the triggers, for pasted text.

This is good news: as triggers are not involved in GUI, we can use a similar approach as noted in #2672-62

> 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).

We can pre-process the text on JS-side (i.e. split it into lines, if the currently focused editor is a widget) and send this to the client-side, via a special message.

#39 - 02/23/2016 01:13 PM - Constantin Asofiei

Constantin Asofiei wrote:

> 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).

> We can pre-process the text on JS-side (i.e. split it into lines, if the currently focused editor is a widget) and send this to the client-side, via a special message.

Constantin Asofiei wrote:

> We can pre-process the text on JS-side (i.e. split it into lines, if the currently focused editor is a widget) and send this to the client-side, via a special message.

Some follow up to this: this will work OK only if the editor is currently empty. I think is better for the client to just "merge" the text into the editor and all its current content pre-processed again (using the approach mentioned above), avoiding the TypeAhead/key processing in P2J.

#40 - 02/24/2016 12:55 AM - Sergey Ivanovskiy

Constantin Asofiei wrote:

> For editor, we need to consider if pasting text into it may or may not result in invoking a trigger. Each character in the pasted text is simulated as a key press in P2J, so we can't easily optimize it if triggers are involved. We need some tests to find how 4GL reacts on PASTE: if there is a trigger on a char in the text, does it get invoked? The following simple tests shows that triggers are not involved when pasting text into the editor:

```
DEF VAR ch AS CHAR VIEW-AS EDITOR SIZE 20 BY 20.
DEF VAR i AS INT.
i = 0.
ON 'a':U ANYWHERE
DO:
   i = i + 1.
   MESSAGE i "a".
   RETURN.
END.

UPDATE ch.
```

> Have some text with "a" chars in the clipboard and paste it into the editor: the trigger will not be executed. But this is only for GUI - the linux ChUI client does invoke the triggers, for pasted text.

I don't know it will be correct if the triggers on the inserted text will be invoked before the text will be drawn. If the text is changed by the triggers, then it will be okay to display the text after the triggers invoked. The triggers are passed, then the transformed text is displayed.

#41 - 02/24/2016 03:00 AM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> I don't know it will be correct if the triggers on the inserted text will be invoked before the text will be drawn. If the text is changed by the triggers, then it will be okay to display the text after the triggers invoked. The triggers are passed, then the transformed text is displayed.

As I mentioned in the note, GUI and ChUI 4GL behave differently:

1. ChUI 4GL does invoke the triggers when pasting text; so our current approach of processing each char via TypeAhead/processKey is correct for pasted text - triggers will be invoked, etc; we don't need to change anything here
2. GUI 4GL does not invoke triggers when pasting text: so we can't use TypeAhead/processKey, to process the text char by char. We need to explicitly "paste" the text into the editor and re-split each line (from the line where the pasted text was added downwards.

#42 - 02/24/2016 03:26 AM - Constantin Asofiei

Actually, the PASTE in GUI 4GL doesn't invoke triggers at all, regardless of the widget. We need a generic approach where:

1. if the widget supports PASTE (i.e. is a EDITOR or FILL-IN), then paste the text into it. For non-char fill-ins (or even char, with explicit format...), we need to find how the pasted text is interpreted.
2. if the widget doesn't support PASTE (is a BUTTON or something else), then just discard the text.

Also, the CTRL-V (or SHIFT-INS) keys need to be processed: any associated trigger is invoked BEFORE pasting the text into the widget; also, if the trigger ends with a RETURN NO-APPLY for CTRL-V/SHIFT-INS, then the text is not pasted.

#43 - 03/08/2016 10:53 AM - Greg Shah

If I understand correctly, the remaining work for this task is for Constantin. Correct?

#44 - 03/08/2016 01:14 PM - Sergey Ivanovskiy

> If I understand correctly, the remaining work for this task is for Constantin. Correct?

Yes if the known two issues with editors and blocking threads are for him.

#45 - 03/08/2016 03:27 PM - Greg Shah

- Assignee changed from Sergey Ivanovskiy to Constantin Asofiei

- % Done changed from 0 to 80

- Status changed from New to WIP

Yes, both these issues will be handled by Constantin.

#46 - 03/22/2016 05:39 PM - Constantin Asofiei

Sergey Ivanovskiy wrote:

> Sergey Ivanovskiy wrote:
>
> > I encountered the thread lock in the case of the large paste data in the editor, the main thread and the web task thread are locked. The scenario is the same as it was in
> > #2956 change-37191. After the paste data has been delivered to the Java side, then the editor continues to send text height and width requests to the JS client and simultaneously we minimize and restore the window. Planning to investigate and to fix it first.

> The main thread and the web worker thread are blocked in the case the main thread that owns the gui driver instance is waiting for the text dimensions while the web worker thread processing the minimize request tries to access the gui driver instance. Thus the web task with the text dimensions is waiting in the tasks queue.

The reason for this deadlock is this: when WebClientProtocol.waitForResult is called, if the thread had a selectWindow call, this lock is not released. A fix would be to let all the callers for the waitForResult release the window lock before and and reacquire it after the waitForResult has completed. But I think the fix is more simple: AbstractGuiDriver.[de]iconifyWindow doesn't need to use selectWindow and releaseWindow, as these just post an event to the queue (which will be picked up by the main thread next time events are processed). Also, the AbstractGuiDriver.[de]iconifyWindow are the only two places where a window lock can be obtained from another thread.

#47 - 04/01/2016 09:21 AM - Greg Shah

I can close this task, right?

If I understand correctly, the paste/editor issues are resolved in task branch 1811t revision 10992.

#48 - 04/01/2016 09:25 AM - Constantin Asofiei

Greg Shah wrote:

> I can close this task, right?

Correct.

> If I understand correctly, the paste/editor issues are resolved in task branch 1811t revision 10992.

Yes

#49 - 04/01/2016 09:38 AM - Greg Shah

- % Done changed from 80 to 100

- Status changed from WIP to Closed

#50 - 11/16/2016 12:12 PM - Greg Shah

- Target version changed from Milestone 12 to GUI Support for a Complex ADM2 App

Files

websocket_config.png	58.5 KB	02/10/2016	Sergey Ivanovskiy
web_worker_1.txt	93.5 KB	02/10/2016	Sergey Ivanovskiy
TestEmbeddedJetty.zip	3.37 KB	02/15/2016	Sergey Ivanovskiy
2967_1.txt	27.7 KB	02/20/2016	Sergey Ivanovskiy