

# Optimization and Performance for Web Developers

JAM343

Adam Stanley @n\_adam\_stanley

Tom Anderson @Noctivagan

February 5, 2013



## Adam

Developer Relations

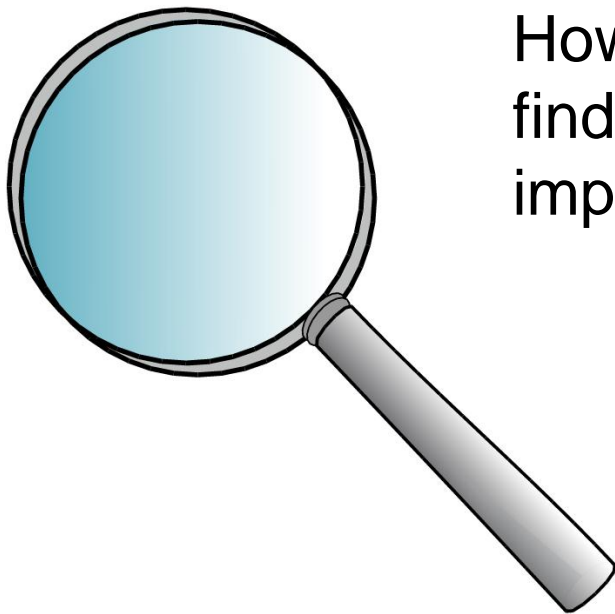


## Tom

Developer Evangelism



Web Inspector will  
not make your  
apps faster.



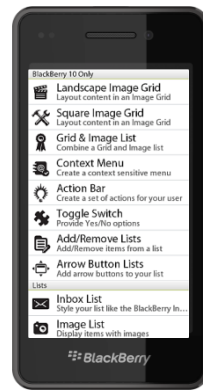
However, it *will* help you  
find opportunities for  
improvement.

But you need to know what to look for ...

- Make fewer HTTP requests for all page resources
- Minimize bandwidth of application resources
- Reduce JavaScript execution time performed by the CPU
- Leverage hardware acceleration wherever possible

# Today's Tasks

- Lab Setup (10 mins)
- Task 1: HTTP requests (20 mins)
- Task 2: Network latency (20 mins)
- Task 3: JavaScript performance (20 mins)
- Task 4: Scrolling and transitions (20 mins)
- Free Time (25 mins)



# Lab setup

Installing sample application

[10 mins]

- Development Tools:
  - ▶ Windows XP, Windows 7 or Mac OS
  - ▶ Chrome Browser
  - ▶ BlackBerry 10 WebWorks SDK
- BlackBerry 10 (Z10 or Dev Alpha)
  - ▶ USB cable
  - ▶ No device? Use the BlackBerry 10 simulator



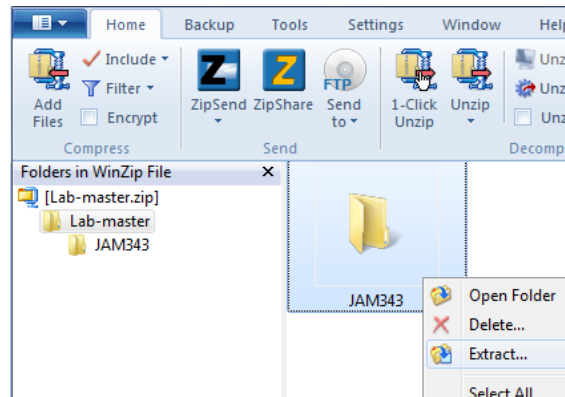
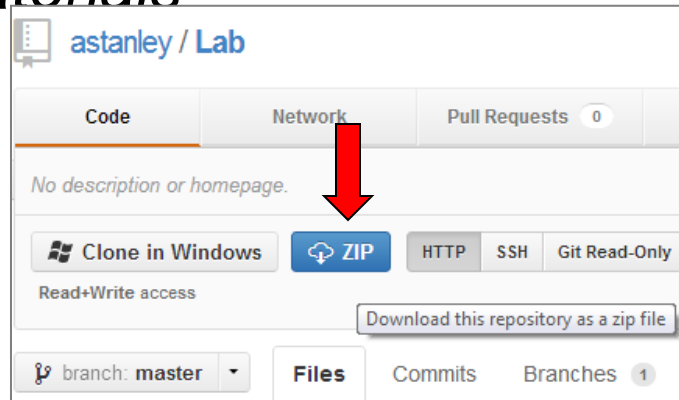
# Lab setup: sample app

- **TASK:** Download and setup Lab materials

- ▶ <https://github.com/astanley/Lab>
- ▶ Click on ZIP button to download

- Extract source files from ZIP

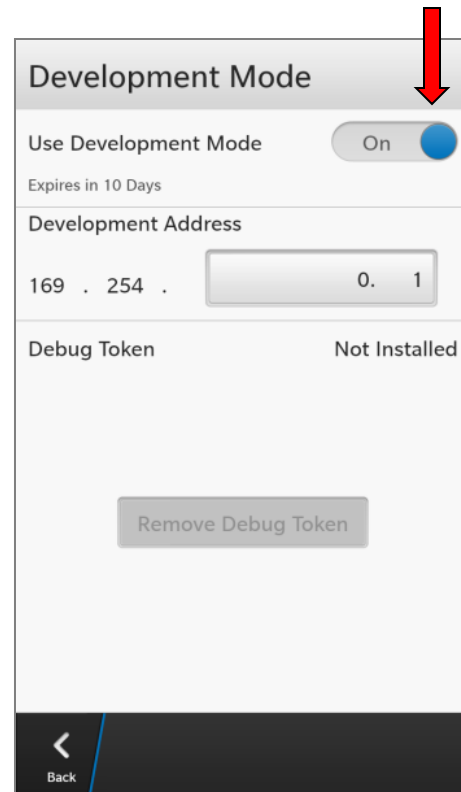
- ▶ Open **Lab-master.ZIP**
- ▶ Extract JAM343 subfolder to C:\Lab
- ▶ Goal = **C:\Lab\JAM343**





# Lab setup: development mode

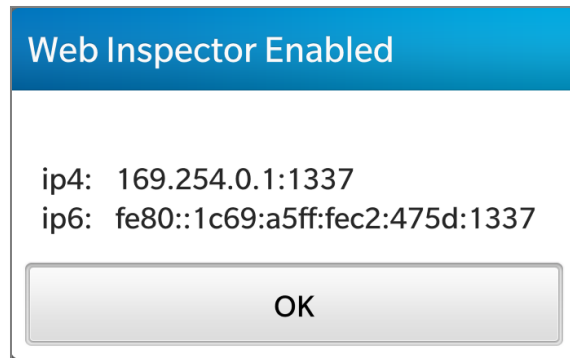
- Enable development mode on your device
  - ▶ Settings → Security and Privacy
  - ▶ Development mode
- Connect your device to USB



- Run loading script from an open command prompt:
  - ▶ Start → Run ... → cmd
  - ▶ cd **C:\Lab\JAM343**
  - ▶ **deploy.bat** [device IP address] [device password]
    - Example: **deploy.bat 169.254.0.1 pass**
- Confirm sample loaded successfully

# Lab setup: remote web inspector

- Start WIC sample application
  - ▶ Should immediately see a dialog
  - ▶ Click OK to enable web inspector
- Open a desktop browser
  - ▶ Browse to <http://169.254.0.1:1337>
  - ▶ Click on “Web Inspector Companion”



# Task 1. HTTP Requests

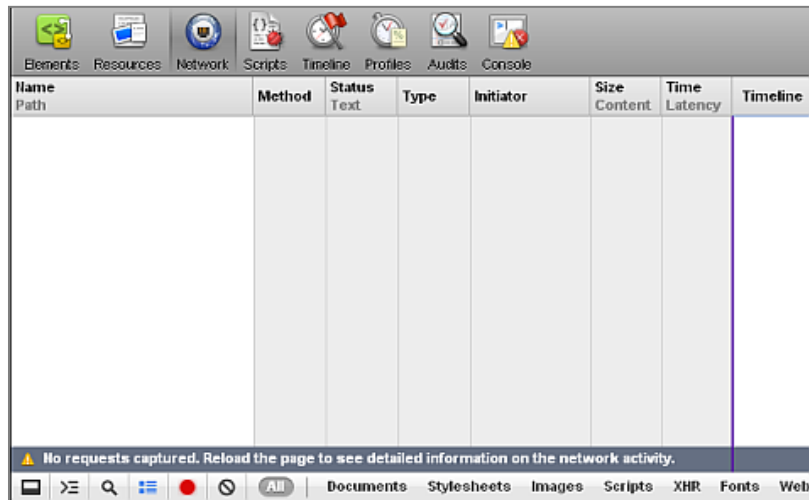
Fewer synchronous requests means faster web apps


[20 mins]

- You may often need to download external resources
  - Examples: images, JavaScript libraries, CSS
- Always try and minimize the total # of separate HTTP requests
- Why?
  - Only a limited number of concurrent connections are allowed
  - Waiting occurs when a web client has to wait for a connection

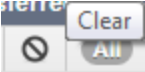
# Quick Lesson: Network Panel

- Setup:
  - ▶ WIC sample application is running
  - ▶ Remote web inspector open <http://169.254.0.1:1337> in Chrome
- Goals: master the timeline
  - ▶ Recording a timeline
  - ▶ View time to download
  - ▶ View resource size
  - ▶ Observe HTTP headers



- ***TASK: Viewing HTTP response headers***
  - ▶ Click on  panel in web inspector
  - ▶ Click on the “Network” link in sample application
  - ▶ Observe Results displayed in network panel of web inspector:
    - How much time did each document requests take to download?
    - What is the HTTP status for the invalid (red) request?
  - ▶ Click on the knight-broken.png resource in web inspector
    - Click on the **Header** pane to see all HTTP headers

# Make fewer HTTP requests

- Tip: Click  button in web inspector to clear results
  - Ensure Network panel in web inspector is opened
- **TASK:** Observe inefficient resource downloading
  - ▶ Click on the “**Display Images**” button in sample application
    - How many HTTP requests are required to display these images?
    - What is the total file size and time required to display all images?
- **TASK:** Observe more efficient resource downloading
  - ▶ Click on the “**Display Image (Sprites)**” button in sample application
    - How many HTTP requests are required to display the same images?



- The Web model is single threaded
  - ▶ Synchronous JavaScript can block UI from being updated
- **TASK:** Observe blocking HTTP requests
  - ▶ Click on the “**Load Content**” button
  - ▶ Use web inspector to measure how long it takes to see new content
- **TASK:** non-Observe blocking HTTP requests
  - ▶ Click on the “**Load Content (async)**” button
  - ▶ Use web inspector to measure how long it takes to see new content

## Task 2. Network Latency

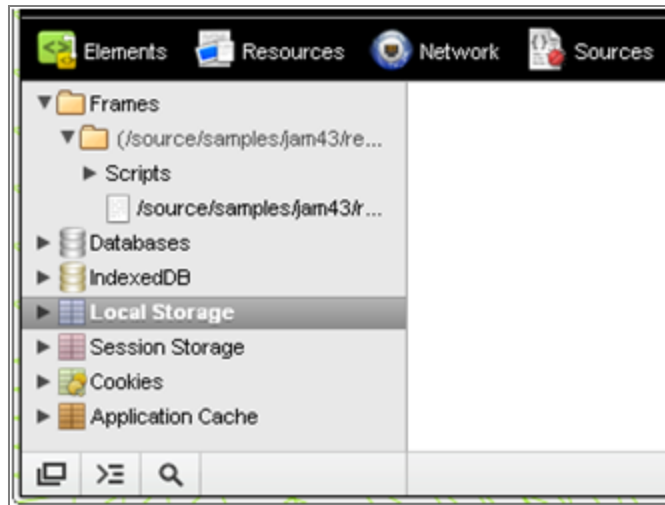
Minimize the amount of bandwidth used

[20 mins]

- Embedding resources and reducing bandwidth
  - ▶ Why download when it is not necessary?
  - ▶ WebWorks allows developers to package resources locally
    - Example: JavaScript, CSS, Images
  - ▶ Minify JavaScript and CSS
    - If you have to download something, make it as small as possible

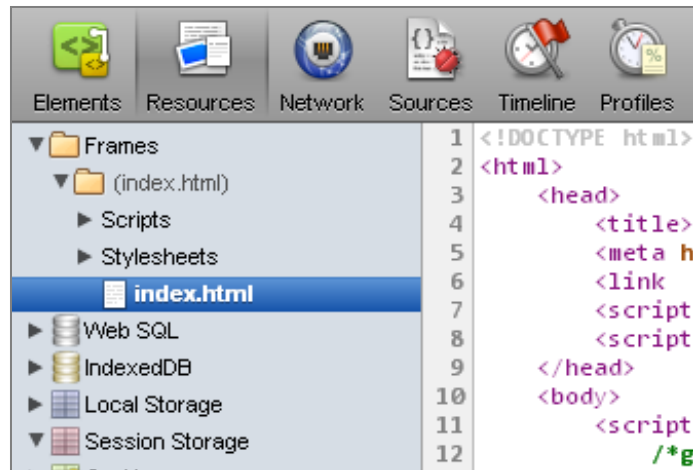
# Quick Lesson: Resources Panel

- Start WIC sample application
  - ▶ Open “Resources” page
- Open <http://169.254.0.1:1337> in Chrome
  - ▶ Click on “Web Inspector Companion”
- Purpose: view active page resources
  - ▶ Page files (examples: html, css, js)
  - ▶ Local storage
  - ▶ Web database
  - ▶ Application cache



# Quick Lesson: Resources Panel

- Viewing the frames tree
  - ▶ View raw page resources
  - ▶ Before they were modified
- ***TASK: Expand Frames tree***
  - ▶ Expand **Scripts** and **Stylesheets**
  - ▶ What JS and CSS frameworks are being used in this app?
  - ▶ Are these local or external resources?



- **TASK:** Analyze minified JavaScript and CSS
  - ▶ Open “Network” page in sample application
  - ▶ Click on Network panel in Web inspector (clear results in advance)
  - ▶ Click on “**Get jQuery**” button in sample application (bottom of page)
  - ▶ How large is jquery-1.9.0.js ?
  - ▶ Click on “**Get jQuery (min)**” button in sample application
  - ▶ How large is jquery-1.9.0.min.js ?
  - ▶ How much quicker does it take to download the minified version of jQuery?

# Task 3. JavaScript

Reducing execution time required by CPU

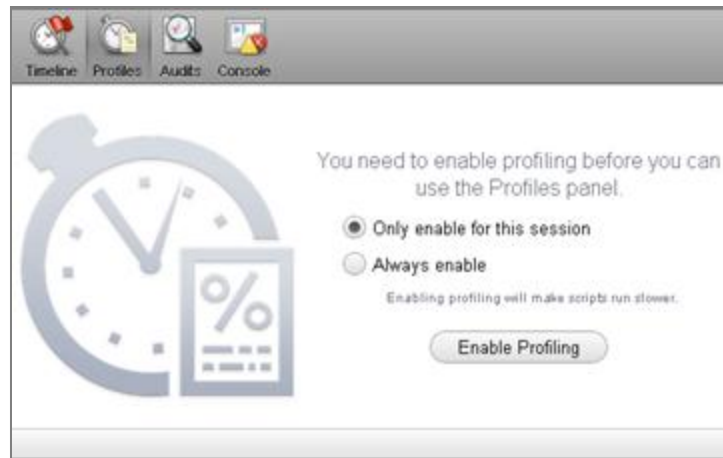
[20 mins]

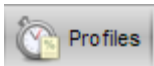
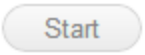

- The web is a single threaded model
  - ▶ The more time spent processing JavaScript means less time doing anything else
  - ▶ Finding inefficient JavaScript
    - Examples: loops, setTimeout, polling
  - ▶ Preventing JavaScript from blocking the UI
    - Asynchronous vs. Synchronous, non-blocking dialogs




# Quick Lesson: Profiles Panel

- Start WIC sample application
  - Open “Profiles” page
- Open <http://169.254.0.1:1337> in Chrome
- What is profile information?
  - Measures JavaScript execution
  - Manually started & stopped
  - Results analyzed in % or milliseconds

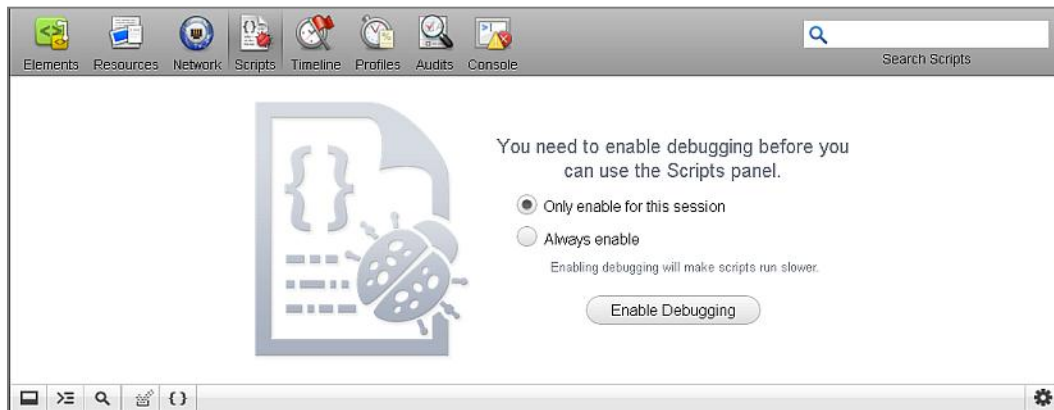


- **TASK:** *Manually collecting JavaScript CPU profile info*
  - ▶ Click on  panel in web inspector
  - ▶ Click  button to begin profiling a session
  - ▶ From the sample app, click **Start 5s Profile Script** button
  - ▶ Once alert box is displayed, click **Stop** button in Web Inspector
  - ▶ Select  from CPU profiles results (left side of panel)
    - Which 3 functions used the most CPU time % during this profile session?
    - How much time did each of these methods run for?

- **TASK:** Finding inefficient JavaScript
  - ▶ Select  result from web inspector
  - ▶ Expand results in the **Function** column to see calling methods
  - ▶ Which parent method called the **updateGraphics** method?
  - ▶ Which line of JavaScript source was this function called from?
  - ▶ How can you easily access the JavaScript source from the profiles panel?


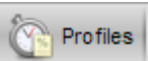
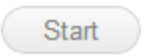
# Quick Lesson: Sources Panel

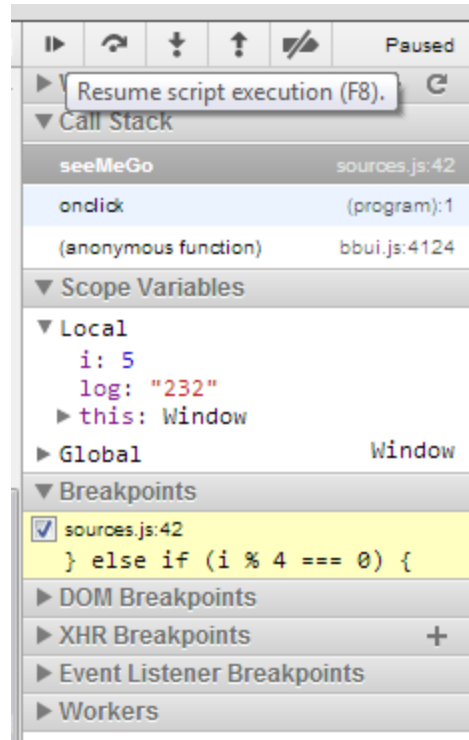
- Pause JavaScript execution at runtime
- Click on Sources panel in web inspector
  - ▶ If prompted, enable debugging



# Quick Lesson: Sources Panel

- ***TASK: Setting a breakpoint***

- ▶ Expand 'Show Navigator' button 
- ▶ Double click on **profiles.js** file
- ▶ Click on line 45 of **profiles.js** to set breakpoint
- ▶ Click on  **Profiles** panel in web inspector
- ▶ Click  button to begin profiling a session
- ▶ Click **Start 5s Profile Script** button in app again
  - Script execution should pause on line 42
- ▶ F10 = step over, F8 = resume execution
- ▶ Click on line 45 again to unset breakpoint



- Preventing JavaScript from blocking the UI
  - Avoid the use of alert() or confirm() dialogs
    - Use WebWorks system dialogs instead
- **TASK:** Comparing different types of JavaScript prompts
  - Open Sources page in sample application
  - Click on the “**Show Time**” button
  - What happens to the time when the alert() dialog is displayed?
  - CTRL + R to reload the application, open Sources page.
  - Click on the “**Show Time (non-blocking)**” button
  - What happens to the time when the toast() dialog is displayed?

# Task 4. Scrolling & transitions

Leverage hardware acceleration when possible

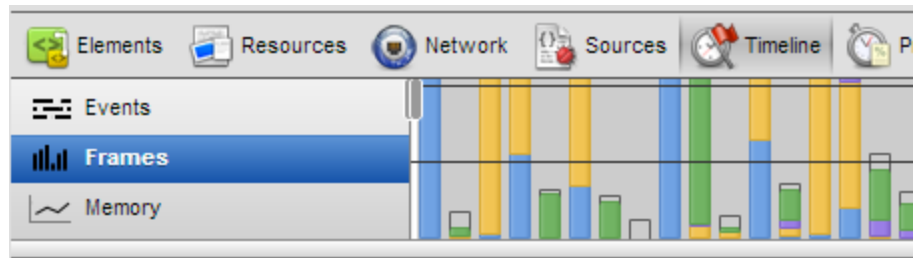
[20 mins]

- Scrolling should be fast!
  - ▶ Poor scrolling performance is a common issue with the Web
    - Solution: Enable hardware acceleration
- Page transitions should be fast!
  - ▶ Developers often use JavaScript for animation – NO!
    - Solution: Use CSS 3D transformations to enable hardware acceleration
    - Tip: AVOID using JavaScript to perform transitions (high CPU load)


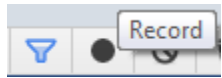

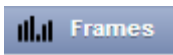


# Quick Lesson: Timeline Panel

- Start WIC sample application
- Open <http://169.254.0.1:1337> in Chrome
- Purpose of Timeline panel:
  - ▶ Page loading events
  - ▶ Frames per second
  - ▶ DOM memory usage



- ***TASK: Read the Frames View***

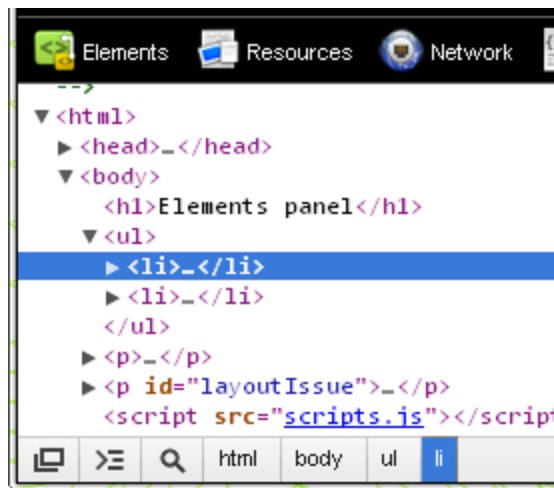
- ▶ Click on the  panel in web inspector
- ▶ Click the  button to start recording the timeline
- ▶ CTRL + R to reload the sample application page
- ▶ Click the  button to stop recording the timeline
- ▶ Click on the  view
- ▶ Observe Results in web inspector (Timeline panel → Frames):
  - What color is used to represent painting to the screen?
  - How many frames did it take to draw the page? Frames per second?

- **TASK:** Observe CPU-powered scrolling performance
  - ▶ Click on the “Timeline” page in sample application
  - ▶ Using the Timeline panel in web inspector, begin recording a session
  - ▶ Find “100 rows of data” section in page
  - ▶ Scroll to the bottom of the 100-rows section
  - ▶ Stop recording the session in web inspector
  - ▶ How many frames are being drawn per second while scrolling?
  - ▶ What is the length of time for each frame?

- **TASK:** Observe GPU-powered scrolling performance
  - ▶ Click on the “Timeline GPU” page in sample application
  - ▶ Using the Timeline panel in web inspector, begin recording a session
  - ▶ Find “100 rows of data” section in page
  - ▶ Scroll to the bottom of the 100-rows section
  - ▶ Stop recording the session in web inspector
  - ▶ How much faster is this style of scrolling?
  - ▶ How many more frames are drawn per second while scrolling?
  - ▶ Next: how do we get such good performance?

# Quick Lesson: Elements Panel

- Start WIC sample application
- Open <http://169.254.0.1:1337> in Chrome
- Purpose:
  - ▶ Interact with the Document Object Model (DOM)
  - ▶ Page markup displayed on the left.
  - ▶ Additional DOM sub panels on the right.



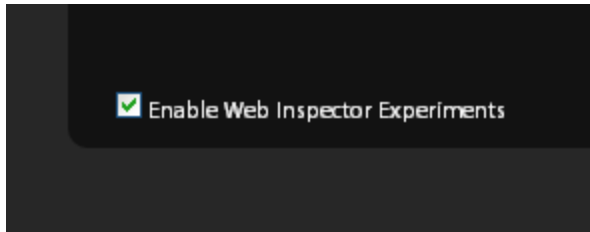
- Live DOM interaction
  - ▶ Open the “Timeline GPU” page in sample application
  - ▶ Click on the Elements panel in web inspector
  - ▶ Expand the markup until you find the `<div id=“scrollcontainer”>` element
- **TASK:** Changing CSS of the live DOM
  - ▶ Expand the Styles view on the right of the Elements panel
  - ▶ What value(s) are displayed in the ‘Matched CSS Rules’ section?
  - ▶ Disable “**-webkit-overflow-scrolling: touch**” checkbox
  - ▶ What happens to scrolling performance when this feature is disabled?

# Free Time: Bonus tasks

Tips and tricks


[25 mins]

- Start WIC sample application
  - ▶ Open “Advanced Features” page
- Open <http://169.254.0.1:1337> in Chrome
  - ▶ Enable **Enable Web Inspector Experiments** checkbox

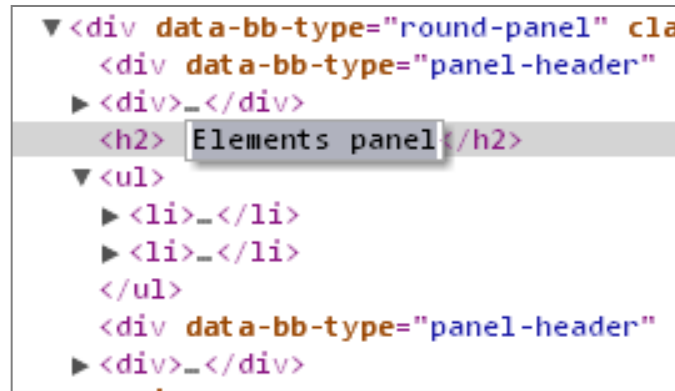


- ▶ Click on “Web Inspector Companion”



- ***TASK: Overriding user agent***
  - ▶ Click **Display user agent** button in the sample application.
    - What user agent value is displayed?
  - ▶ Open **Settings** screen in web inspector 
  - ▶ Select the **Overrides** tab
  - ▶ Click **User Agent** checkbox
  - ▶ Select a different option user agent drop-down box
  - ▶ Click **Display user agent** button again in the sample application
    - What new user agent value is displayed?

- Live DOM editing
  - ▶ Very useful for rapid UI testing
  - ▶ Tip: expand or collapse any elements



```
▼ <div data-bb-type="round-panel" cla
  <div data-bb-type="panel-header"
    ▶ <div>_</div>
    <h2>Elements panel</h2>
    ▼ <ul>
      ▶ <li>_</li>
      ▶ <li>_</li>
    </ul>
    <div data-bb-type="panel-header"
      ▶ <div>_</div>
```

- ***TASK: Modify page content***
  - ▶ Open “Elements” page in sample application
  - ▶ Open Elements panel in web inspector
  - ▶ Double click DOM elements in the markup view.
  - ▶ Change the contents of an element and press ENTER.
    - E.g. `<h2>Elements Panel</h2>` → `<h2>Let's rock and roll this!</h2>`

# Bonus Tasks: Resources panels

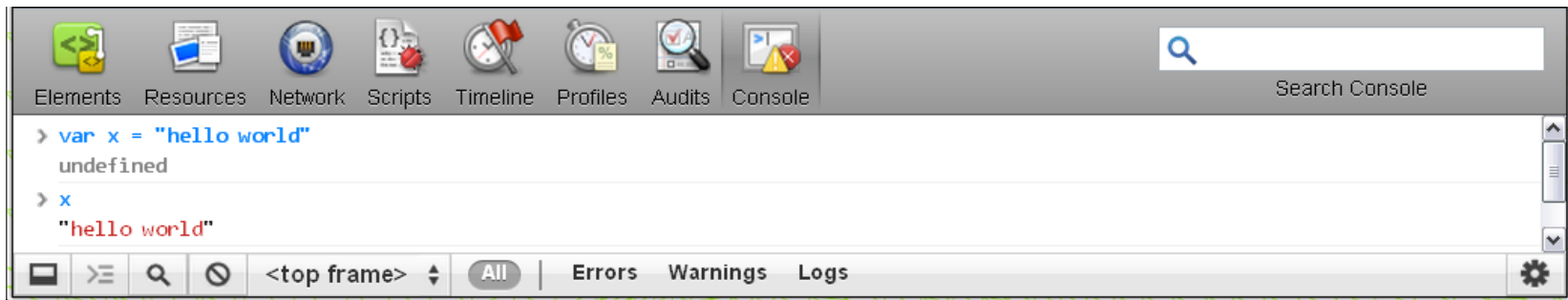
- Interacting with Local storage
  - ▶ Enables offline application data
- ***TASK: Viewing and modifying data***
  - ▶ Open Resources page in sample application
  - ▶ In web inspector, click on **Local Storage** item in resources panel
  - ▶ From sample application, click **Insert into localStorage** button
    - What happens to local storage results in resources panel?
  - ▶ In web inspector, double click key or value, modify values
    - E.g. “Foo” → “Bacon”
  - ▶ Right click a local storage record and select **Delete**



The screenshot shows a web interface titled "Local Storage". It contains a label "Item value:" followed by a text input field containing the text "Foo". Below the input field is a button labeled "Insert into localStorage". At the bottom of the interface, it says "There are 1 item(s) saved in Local Storage." followed by a button labeled "Clear localStorage".

# Bonus Tasks: Console Panel

- Setup:
  - ▶ WIC sample application is running
  - ▶ Remote web inspector open <http://169.254.0.1:1337> in Chrome



- Running JavaScript from the console
  - ▶ You can access any variable or method that the web page can.
- ***TASK: Viewing runtime data***
  - ▶ Open Console panel in Web inspector
  - ▶ What do you see when you type **window.localStorage** or **document.body** and press enter?
- ***TASK: Manually running JavaScript***
  - ▶ Type **window.location.reload()** to reload the current page.
  - ▶ What happens when you type **openBrowser()**?

- Sessions:

- ▶ JAM333 – Porting Apps: Apache Cordova
- ▶ JAM343 – Lab: Optimization and Performance
- ▶ JAM360 – HTML5 frameworks: Up-And-Comers
- ▶ JAM397 – A pixel is not a pixel
- ▶ JAM336 – Web API Deep Dive: Leveraging PIM

- Resources:

- ▶ <http://developer.blackberry.com/html5>
- ▶ <http://github.com/blackberry>

# THANK YOU

JAM343

Adam Stanley @n\_adam\_stanley

Tom Anderson @Noctivagan

February 5, 2013