WearScript Documentation

Release .0.1.0

Brandyn White

April 27, 2014

WearScript combines the power of Android development on Glass with the learning curve of a website. Go from concept to demo in a fraction of the time. For an overview check out the intro video and sample script below. Visit https://github.com/wearscript for the goods.

One-Line Installer(Linux/OSX): Execute the following in a shell to install WearScript on your Glass and authenticate with our default server.

```
curl -L http://goo.gl/nRjW6y > install.py && python install.py
```

```
// Sample WearScript
<html style="width:100%; height:100%; overflow:hidden">
<body style="width:100%; height:100%; overflow:hidden; margin:0">
<canvas id="canvas" width="640" height="360" style="display:block"></canvas>
<script>
function cb(data) {  // Changes canvas color depending on head rotation
    if (data['type'] == WS.sensor('orientation')) {
        ctx.fillStyle = 'hsl(' + data['values'][0] + ', 90%, 50%)'
        ctx.fillRect(0, 0, 640, 360);
    }
}
function server() {
    WS.log('Welcome to WearScript');  // Write to Android Log and Playground console
    WS.say('Welcome to WearScript');  // Text-to-speech
    // Stream camera images and all sensors to the WearScript Playground Webapp
    var sensors = ['gps', 'accelerometer', 'magneticField', 'orientation', 'gyroscope',
                   'light', 'gravity', 'linearAcceleration', 'rotationVector'];
    for (var i = 0; i < sensors.length; i++)
        WS.sensorOn(WS.sensor(sensors[i]), .15, 'cb');
    WS.cameraOn(2);
    WS.dataLog(false, true, .15);
}
function main() {
    if (WS.scriptVersion(0)) return;
    ctx = document.getElementById('canvas').getContext("2d");
    WS.serverConnect('{{WSUrl}}', 'server');
}
window.onload = main;
</script></body></html>
```

CHAPTER 1

Getting Started

You need to install WearScript on Glass and connect it to a server.

- Put WearScript on Glass...
  1. as easy as possible: curl -L http://goo.gl/nRjW6y > install.py && python install.py
  2. and be able to hack the Java code: Follow the *Client Setup (source)*.
- Use our WearScript server: Follow *0a. Using Our Server*
- Run your own WearScript server...
  1. on an existing machine: *0b. Install Server*

CHAPTER 2

Client Setup (source)

2.1 0: Get WearScript Source

1. with one click: Click to Download ZIP
2. using git: git clone https://github.com/wearscript/wearscript-android.git

2.2 1: Setup Your Device

- Put the device in debug mode (see "Turning on debug mode" here https://developers.google.com/glass/gdk)
- Connect your Glass to your computer with a USB cable

2.3 2: Install Client

2.4 2: Get Android Studio and Install Client

First we install/setup Android Studio

- Download/unpack the canary version of Android Studio with Android SDK bundle http://tools.android.com/download/studio/canary/0-3-2 note that this is the latest version WITH the sdk bundled, you can update once it is installed but this simplifies the install
- Locate the "sdk/platform-tools" and "sdk/tools" directories and add them to your system path (this lets you use the "adb and android" commands)
- (OSX): echo "export PATH=$PATH:/Applications/AndroidStudio.app/sdk/platform-tools/:/Applications/AndroidStudio.app/sdk/tools/" >> ~/.profile
- (Windows) from the command prompt(Without quotes) "set PATH=%PATH%;C:YourFolderPathwhereadbIsLocatedsdkplatform-tools"
- NOTE: If it can't find the SDK then follow these steps
  - If you need a new version of the SDK get the ADT Bundle here http://developer.android.com/sdk/index.html
  - Open Android Studio, click on Configure->Project Defaults->Project Structure and under Project SDK click New...->Android SDK and select the "sdk" folder inside of the ADT Bundle

  - If you have changed your SDK path you may need to remove local.properties (it retains the path to use for the project, it'll be reset to default on import)

Now we build/install the client

- The gdk.jar is already included for you in the libraries folder. The steps we ran to get it are: run the "android" command, under Android 4.0.3 (API 15) install Glass Developer Sneak Peek to get the extra library (see https://developers.google.com/glass/develop/gdk/quick-start) and then copy the gdk.jar into the WearScript/libs folder.
- In the WearScript source go to thirdparty and run the command "bash install.sh"
- Start Android Studio (Linux: bash android-studio/bin/studio.sh)
- Click "Import Project" and select wearscript/glass (NOTE: make sure you use the "wearscript/glass" directory, if you select "wearscript" it won't work)
- Select "Import project from external model" and use Gradle
- Select "Use default gradle wrapper" (it is the default), after this it will say Building 'glass' and take some time
- Build using Run->Run 'WearScript'
- After it is built, Select your device and install.
- Keep the screen on while it is installing or it will be started in the background.

2.5 3: Configure Device for Server

- WearScript allows you to control Glass via a WebApp, to do this you have to select a server (see *Server Setup*)
- Go to the server in Chrome (please use Chrome, it is easier for us to support)
- Click "authenticate", then sign-in using your Google acccount (if you are using our server you must be whitelisted before continuing)
- Click QR, then either
  - Paste the adb command while Glass is plugged connected to USB
  - Select WearScript (setup) and scan the QR code (the first time you start the barcode scanner it has a setup menu, just downswipe to dismiss it)

2.6 4: Starting the Client

While you have the webapp open, start the client using one of the following methods * To start with adb use "adb shell am start -n com.dappervision.wearscript/.MainActivity" * To start with Android Studio after the project has been imported (see Install Client (Source)) select Run->Run 'WearScript'. * To start with "Ok Glass" say "wear a script"

CHAPTER 3

Server Setup

3.1 0a. Using Our Server

- Visit https://api.picar.us/wearscript/, then sign-in using your Google acccount
- Alternatively, we provide a server that tracks the 'dev' branch https://api.picar.us/wearscriptdev/

3.2 0b. Install Server

- Server code is located at https://github.com/wearscript/wearscript-server.git
- Playground webapp is located at https://github.com/wearscript/wearscript-playground.git (you need both)
- Linux is highly recommended, we have not tested this on OSX or Windows (feel free to try)
- A few "alternate" options are listed below that may be useful if you run into problems
- Tested on Ubuntu 13.04, if you want support it helps if you stick with this or a new Ubuntu if possible.
- Ubuntu packages: apt-get install golang git mercurial redis-server
- Setup the config.go file (look at config.go.example)
- Run /server/install.sh (this does basic dependencies and such)
- Start with ./server and continue with "Connecting the Client to the Server"

3.3 Alternate: Installing Go (manually)

- wget https://go.googlecode.com/files/go1.1.1.linux-amd64.tar.gz
- tar -xzf go1.1.1.linux-amd64.tar.gz
- Put "export GOROOT=<yourpath>/go" and "export GOPATH=<yourpath>/gocode" in your .bashrc
- The "gocode" is where packages will be stored and "go" is the location of the extracted folder.

3.4 Alternate: Install Redis

- Follow instructions here http://redis.io/download (tested on 2.6.*)

CHAPTER 4

GDK Cards

WearScript uses an abstraction called a CardTree that allows for a hierarchy of cards where a node can optionally have either a menu or another set of cards beneath it and every card can have a tap/select callback. The syntax is overloaded to make common functionality concise.

The following displays a GDK card (consists of a body and footer) .. code-block:: javascript

> var tree = new WS.Cards(); tree.add('Body text', 'Footer text'); WS.cardTree(tree); WS.displayCardTree();

Lets add another card .. code-block:: javascript

> var tree = new WS.Cards(); tree.add('Body 0', 'Footer 0'); tree.add('Body 1', 'Footer 1'); WS.cardTree(tree); WS.displayCardTree();

Select and tap callbacks are optional arguments .. code-block:: javascript

> var tree = new WS.Cards(); tree.add('Body 0', 'Footer 0', function () {WS.say('Selected')}); tree.add('Body 1', 'Footer 1', undefined, function () {WS.say('Tapped')}); WS.cardTree(tree); WS.displayCardTree();

A menu is added with alternating title and callback arguments at the end of the parameter list. Tap/select parameters (if present) precede them. .. code-block:: javascript

> var tree = new WS.Cards(); tree.add('Body 0', 'Footer 0', 'Say 0', function () {WS.say('Say 0')}, 'Say 1', function () {WS.say('Say 1')}); tree.add('Body 1', 'Footer 1', function () {WS.say('Selected')}, 'Say 0', function () {WS.say('Say 0')}, 'Say 1', function () {WS.say('Say 1')}); tree.add('Body 2', 'Footer 2', function () {WS.say('Selected')}, function () {WS.say('Tapped')}, 'Say 0', function () {WS.say('Say 0')}, 'Say 1', function () {WS.say('Say 1')}); WS.cardTree(tree); WS.displayCardTree();

A subtree of cards is added by creating another set of cards and placing it as the last parameter (may only have a menu or a subtree for a card). There is no depth limit for subtrees.

```
var tree = new WS.Cards();
tree.add('Body 0', 'Footer 0');
var subtree = new WS.Cards();
subtree.add('Sub Body 0', 'Sub Footer 0');
subtree.add('Sub Body 1', 'Sub Footer 1');
tree.add('Body 1', 'Footer 1', subtree);
WS.cardTree(tree);
WS.displayCardTree();
```

CHAPTER 5

Tips/Tricks

- If you swipe down the script will continue to run in the background
- To turn off WearScript start it and select “Stop” (swipe one option to the right)
- When calling WS.serverConnect, if the argument passed is exactly ‘{{WSUrl}}’ then it will be replaced with the websocket url corresponding to the server the playground is running on and the last QR code generated.
- Unless you use a script that makes a server connection (i.e., WS.serverConnect(‘{{WSUrl}}’), ‘callback’) you will not be able to control WearScript from the webapp
- More interesting uses of WS.serverConnect include making a custom server for your application and then Glass will connect to it and stream data while your app can continue to control Glass.
- Every time you press the QR button on the webapp you get a unique auth key which replaces the previous.
- Multiple Glass devices can use the same QR code
- You only need to auth your Glass once and then again anytime you want to change servers (using the adb command provided when you press the QR button).
- When using scripts in the Playground editor, make sure to specify http:// or https:// and NOT use refer to links like <script type=”text/javascript” src=”//example.com/test.js”></script>. The script you put in the editor will be saved locally on Glass, and links of that form will not work.
- If you are connected to a server and use WS.log(‘Hi!’), that message will show up in the Android logs and the javascript console in the Playground.

CHAPTER 6

Hacking WearScript

6.1 Code Organization

- Android (Phone + Glass): https://github.com/wearscript/wearscript-android.git
- Server: https://github.com/wearscript/wearscript-server.git
- Playground Webapp: https://github.com/wearscript/wearscript-playground.git

6.2 Travis-CI

The current test setup just builds the server after each commit.

6.3 Resources

These are helpful for development

- https://developers.google.com/glass/policies
- https://code.google.com/p/google-api-go-client/source/browse/mirror/v1/mirror-gen.go
- http://golang.org/doc/effective_go.html
- https://developers.google.com/glass/playground

CHAPTER 7

API Reference

In your Javascript environment, an object *WS* is initialized and injected for you with the following methods.

7.1 General

scriptVersion(int version) [boolean] Checks if the webview is running on a specific version.

sensorOn(int type, double period, [String callback]) [void] Turn on the sensor and produce data no faster than the specific period. Best used with WS.sensor like WS.sensorOn(WS.sensor('light'), .5). Optional callback name that is called at most once per period.

- Sensor values
- For the Android built in sensors see the Android docs for their values, custom values are...
- battery: Values [battery_percentage] (same as displayed in the Glass settings)
- pupil: Values [pupil_y, pupil_x, radius]
- gps: Values [lat, lon]
- Callback has parameters of the form function callback(data) {} with "data" being an object of the form property(value type) of...
- name(string): Unique sensor name (uses Android name if one exists)
- type(int): Unique sensor type (uses Android type if one exists), convert between them using WS.sensor(name) -> type
- timestamp(double): Epoch seconds from when we get the sensor sample (use this instead of Raw unless you know better)
- timestampRaw(long): Potentially differs per sensor (we use what they give us if available), but currently all but the light sensor are nanosec from device uptime
- values(double[]): Array of float values (see WS.sensor docs for description)

sensorOff(int type) [void] Turns off sensor

say(String message) [void] Uses Text-to-Speach to read text

qr(String callback) [void] Open a QR scanner, return scan results via a callback from zxing

- Callback has parameters of the form function callback(data, format)

- data(string): The scanned data (e.g., http://wearscript.com) base64 encoded (e.g., aHR0cDovL3dlYXJzY3JpcHQuY29t) as a security precaution. Decode by doing atob(data) in javascript.
- format(string): The format of the data (e.g., QR_CODE)

log(String message) [void] Log a message to the Android log and the JavaScript console of the webapp (if connected to a server).

displayWebView() [void] Display the WebView activity (this is the default, reserved for future use when we may have alternate views).

shutdown() [void] Shuts down wearscript

data(int type, String name, String valuesJSON) [void] Log "fake" sensor data made inside the script, will be logged based on the WS.dataLog settings.

audioOn() [void] Logs noise level to server

audioOff() [void] Stops logging noise

cameraOn(double period) [void] Camera frames are output based on the *cameraCallback* and *dataLog* options.

cameraPhoto() [void] Take a picture and save to the SD card.

cameraVideo() [void] Record a video and save to the SD card.

cameraCallback(int type, String callback) [void] Type 0=local camera, 1=remote camera (subject to change).

- Callback has parameters of the form function *callback(String imageb64)*
- imageb64 being the image represented as a base64 encoded jpeg

cameraOff() [void] Turns off camera

activityCreate() [void] Creates a new activity in the foreground and replaces any existing activity (useful for bringing window to the foreground)

activityDestroy() [void] Destroys the current activity.

wifiOn([String callback]) [void] Turn on wifi scanning with optional callback

- Callback has parameters of the form function callback(results) where results is an array of objects each of the form following ScanResult except for the timestamp...
- timestamp(double): Epoch seconds from when we get the wifi scan
- capabilities(string): Authentication, key management, and encryption schemes supported by the access point
- SSID(string): network name
- BSSID(string): address of the access point
- level(int): detected signal level in dBm (may have a different interpretation on Glass)
- frequency(int): frequency in MHz of the channel over which the client is communicating with the access point

wifiScan() [void] Request a wifi scan.

wifiOff() [void] Turn off wifi

serverConnect(String server, String callback) [void] Connects to the WearScript server, if given '{{WSUrl}}' as the server it will substitute the user configured server. Some commands require a server connection.

- Callback takes no parameters and is called when a connection is made, if there is a reconnection it will be called again.

serverTimeline(JSON timelineItem) [void] If connected to a server, has that server insert the timeline item (exact mirror timeline item syntax serialized to JSON)

dataLog(boolean local, boolean server, double sensorPeriod) [void] Log data local and/or remote, buffering sensor packets according to sensorPeriod.

wake() [void] Wake the screen if it is off, shows whatever was there before (good in combination with WS.activityCreate() to bring it forward).

sound(String type) [void] Play a stock sound on Glass. One of TAP, DISALLOWED, DISMISSED, ERROR, SELECTED, SUCCESS.

publish(String channel, args[]) [void] Sends PubSub messages to other devices

subscribe(String channel, Function callback) [void] Receives PubSub messages from other devices. Callback is provided the data expanded (e.g., if ['testchan', 1] is received then callback('testchan', 1) is called). Using javascript's 'arguments' functionality to get variable length arguments easily.

7.2 GDK-only

gestureCallback(String event, String callback) [void] Register to get gesture events using the string of one of the events below (following GDK names, see below).

- Each of these follows the parameters provided by the GDK
- onGesture(String gesture): The gestures that can be returned are listed here: LONG_PRESS, SWIPE_DOWN, SWIPE_LEFT, SWIPE_RIGHT, TAP, THREE_LONG_PRESS, THREE_TAP, TWO_LONG_PRESS, TWO_SWIPE_RIGHT, TWO_SWIPE_UP, TWO_TAP
- onFingerCountChanged(int previousCount, int currentCount):
- onScroll(float displacement, float delta, float velocity):
- onTwoFingerScroll(float displacement, float delta, float velocity):

speechRecognize(String prompt, String callback) [void] Displays the prompt and calls your callback with the recognized speech as a string

- Callback has parameters of the form function *callback(String recognizedText)*

liveCardCreate(boolean nonSilent, double period) [void] Creates a live card of your activity, if nonSilent is true then the live card is given focus. Live cards are updated by polling the current activity, creating a rendering, and drawing on the card. The poll rate is set by the period. Live cards can be clicked to open a menu that allows for opening the activity or closing it.

liveCardDestroy() [void] Destroys the live card.

cardFactory(String text, String info) [JSON] Creates a cardJSON that can be given to the card insert/modify functions, the "text" is the body and the "info" is the footer.

cardInsert(int position, JSON card) [void] Insert a card at the selected position index.

cardDelete(int position) [void] Delete a card at the selected position index.

cardModify(int position, JSON card) [void] Modify (replaces) a card at the selected position index.

cardCallback(String event, String callback) [void] Register to get card callback events using hte string of one of the events below (following GDK names, see below).

- Each of these follows the callbacks of the same name in the GDK
- onItemClick(int position, int id): Called when a card is clicked

- onItemSelected (int position, int id): Called when a card is displayed
- onNothingSelected(): Called when not on a card (e.g., scrolling between cards or when there are no cards).

displayCardScroll() [void] Displays the card scroll view instead of the webview.

7.3 Sensor Types

Sensors have unique names and integer types that are used internally and can be used as WS.sensor('light') which returns 5. The standard Android sensor types are positive and custom types are given negative numbers.

- pupil: -2
- gps: -1
- accelerometer: 1
- magneticField: 2
- orientation: 3
- gyroscope: 4
- light: 5
- gravity: 9
- linearAcceleration: 10
- rotationVector: 11

CHAPTER 8

Websocket Wire Format

All encoding is done using msgpack with lists as the base structure that have a string type as the first element. Websockets with a binary message payload a re used. Message delivery follows a pub/sub pattern.

8.1 Motivation and Goals

- Keep local data local if possible (e.g., glass -> phone shouldn't need to go through a server if they are connected)
- Pub/Sub over Point-to-Point: Focusing on channels which represent data types naturally handles 0-to-many devices reacting to it.
- Minimize data transfer, if nothing is listening on a channel then sending to it is a null-op
- Support a directed-acyclic-graph topology, where local devices can act as hubs (e.g., a phone is a hub for a watch, Glass, and arduino) and connect to other hubs (e.g., a remote s

erver) * Instead of having strict guarantees about delivery, provide a set of constraints that can be met given the fickle nature of mobile connectivity to eliminate edge cases

8.2 Protocol Rules

- Messages are delivered with "best effort" but not guaranteed: devices can pop in and out of existance and buffers are not infinite
- If a sender's messages are delivered they are in order
- A message SHOULD only be sent to a client that is subscribed to it or is connected to clients that are
- A message sent to a client that neither it or its clients are subscribed to MUST ignore it
- The "subscriptions" channel is special in that it MUST be sent to all connected clients
- When a client connects to a server it MUST be send the subscriptions for itself and all other clients
- If multiple channels match for a client the most specific MUST be called and no others
- When a client subscribes to a channel a single callback MUST be provided

The following table gives examples for when data will be sent given that there is a listener for the specified channel.

sub	send	sent
“”	a	false
“”	“”	true
a	a	true
a	a:b	true
b	a:b	false
a:	a	false
a:	a:b	false
a:	a::b	true
a:b	a	false
a:b	a:b	true
a:b	a:bc	false
a:b	a:b:c	true

CHAPTER 9

Contributing

- All patches must be provided under the Apache 2.0 license
- Please use the -s option in git to "sign off" that the commit is your work and you are providing it under the Apache 2.0 license
- Submit a Github pull request to the "dev" branch and ideally discuss the changes with us in IRC
- We will look at the patch, test it out, and give you feedback
- New features should generally be put in feature branches
- Avoid doing minor whitespace changes, renamings, etc. along with merged content. These will be done by the maintainers from time to time but they can complicate merges and should be done seperately.
- All pull requests should be "fast forward"
  - If there are commits after yours use "git rebase -i <new_head_branch>"
  - If you have local changes you may need to use "git stash"
  - For git help see progit which is an awesome (and free) book on git

CHAPTER 10

Eye Tracking

We developed a custom eye tracker for Glass so that we can experiment with using it as an input device. It is for research and development purposes, there is a lot of potential for this sub-project but it is still early so bare with us. To make it easier to support you, please follow *vm-setup*. See the video below for details of this project.

10.1 Getting Started

- Install the VM (*vm-setup*)
- Acquire/build an eye tracker (best to contact brandyn in IRC)
- Attach the webcam to the vm (see below)
- In hardware/eyetracker/ run python track.py debug, that will show you the eye camera and it should show a ring around your pupil
- We have several things you can do with it after this, but we are tidying things up (more info will be posted here after)

10.2 Building an Eye Tracker

- http://www.amazon.com/Microsoft-LifeCam-HD-6000-Webcam-Notebooks/dp/B00372567A
- 2 LEDs (recommend you get 4+ as you may break/lose some while soldering): http://www.digikey.com/product-detail/en/SFH%204050-Z/475-2864-1-ND/2207282 (shipping was $2.50 and each LED is ~$.75-$1 depending on amount)
- Access to a 3d printer (print the .stl file, currently #2 is the best design)
- Skills: Need to teardown a camera (requires patience), surface mount (de)solder LEDs (is doable with normal soldering equipment), need to take off/break the IR filter on the optics (requires the care)
- Tools (see build video): Side cutter, thin phillips screwdriver, spudger/x-acto knife (remove IR filter), soldering iron, solder, prybar, wrench (for cracking open case)

10.3 Attach the Webcam to the VM

- From a terminal on the host OS
  - VBoxManage list webcams

  - VBoxManage list vms
  - VBoxManage controlvm <yourvm name> webcam attach <your webcam id>
- From the Virtual Box gui (when vb.gui = true in Vagrantfile)
  - Devices -> Webcams -> Microsoft LifeCam HD-6000 for Notebooks
  - If it isn't listed or otherwise isn't working try rebooting the box or using a different usb port with the camera

10.4 Tips

- The webcam must be manually focused, if it appears blurry twist the lens until the image is in focus. It may help to take the webcam out of the plastic mount and hold it roughly where it would be while doing this.
- When using the VM, we want virtual box to use the webcam as a webcam. Virtualbox uses the host to capture from the webcam and has a good driver on the guest side. Specifically we don't want it to expose it through usb directly.
- The 'Vagrantfile' has a line that specifies vb.gui, if this is set to true then the virtual box gui is presented (useful for more easily connecting/disconnecting the webcam). If it is false then we can only use vagrant ssh to connect (though X11 connections are still tunneled). If you change it use "vagrant reload" to restart the vm and parse that file.

CHAPTER 11

About

- OpenShades (the new OpenGlass) is our community
- IRC freenode #wearscript and #openshades (if you want to collaborate or chat that's the place to be)
- Project Lead: Brandyn White (bwhite dappervision com)
- G+ Community (we post work in progress here)
- Youtube (all demo videos)
- Dapper Vision, Inc. (by Brandyn and Andrew) is the sponsor of this project
- Code is licensed under Apache 2.0 unless otherwise specified

CHAPTER 12

Contributors

See contributors for details. Name (irc nick)

- Brandyn White (brandyn)
- Andrew Miller (amiller)
- Scott Greenwald (swgreen_mit)
- Kurtis Nelson (kurtisnelson)
- Conner Brooks (connerb)
- Justin Chase (jujuman)
- Alexander Conroy (geilt)