# SAGE2 Documentation

June 23, 2014

1. [SAGE2](#sage2)  
1.1. [Dependencies](#dependencies)  
1.2. [Install](#install)  
1.3. [Configure](#configure)  
1.4. [Run](#run)  
1.5. [Supported File Types](#supportedfiletypes)  
1.6. [Notice](#notice)  
2. [Install for Mac OSX (10.7 - 10.9)](#installformacosx10.7-10.9)  
2.1. [Download](#download)  
2.2. [Install Dependencies](#installdependencies)  
2.3. [Clone SAGE2](#clonesage2)  
2.4. [Generate HTTPS Keys](#generatehttpskeys)  
2.5. [Install Node.js Modules](#installnode.jsmodules)  
3. [Install for Ubuntu (14.04)](#installforubuntu14.04)  
3.1. [Download](#download-1)  
3.2. [Install Dependencies](#installdependencies-1)  
3.3. [Install Node.js](#installnode.js)  
3.4. [Install FFMpeg](#installffmpeg)  
3.5. [Clone SAGE2](#clonesage2-1)  
3.6. [Generate HTTPS Keys](#generatehttpskeys-1)  
3.7. [Install Node.js Modules](#installnode.jsmodules-1)  
4. [Install for Windows (7 or 8)](#installforwindows7or8)  
4.1. [Download](#download-2)  
4.2. [Install FFMpeg](#installffmpeg-1)  
4.3. [Install Poppler](#installpoppler)  
4.4. [Set Environment](#setenvironment)  
4.5. [Clone SAGE2](#clonesage2-2)  
4.6. [Generate HTTPS Keys](#generatehttpskeys-2)  
4.7. [Install Node.js Modules](#installnode.jsmodules-2)  
5. [Install for Linux openSUSE (13.1)](#installforlinuxopensuse13.1)  
5.1. [Install Dependencies ##](#installdependencies##)  
5.2. [Clone SAGE2 ##](#clonesage2##)  
5.3. [Generate HTTPS Keys ##](#generatehttpskeys##)  
5.4. [Install Node.js Modules ##](#installnode.jsmodules##)  
6. [SAGE2 config](#sage2config)  
6.1. [Fields](#fields)  
6.2. [Sample](#sample)  
6.3. [Note](#note)  
7. [Starting SAGE2 with one button](#startingsage2withonebutton)  
7.1. [Windows](#windows)  
7.2. [Mac OSX](#macosx)  
7.3. [openSUSE](#opensuse)  
7.4. [Ubuntu](#ubuntu)  
8. [SAGE2 Application Methods](#sage2applicationmethods)  
8.1. [Boiler-plate for a canvas application](#boiler-plateforacanvasapplication)  
8.2. [Function Descriptions](#functiondescriptions)  
8.2.1. [Construct](#construct)  
8.2.2. [Init](#init)  
8.2.3. [Draw](#draw)  
8.2.4. [Resize](#resize)  
8.2.5. [Load](#load)  
8.2.6. [Event](#event)  
8.3. [External Libraries](#externallibraries)  
8.4. [How to write an app](#howtowriteanapp)  
8.4.1. [instructions.json](#instructions.json)  
8.4.2. [load into app library](#loadintoapplibrary)  
8.5. [Future: Widgets](#future:widgets)  
8.6. [Future: Inter-application communication](#future:inter-applicationcommunication)  
8.7. [Example Application: Clock](#exampleapplication:clock)

# 1. SAGE2

Browser based implementation of SAGE. A cluster-based html viewer used for displaying elements across multiple browser windows.

## 1.1. Dependencies

* node.js
* ffmpeg
* poppler
* imagemagick

## 1.2. Install

* [Windows Install](https://bitbucket.org/sage2/sage2/wiki/Install%20(Windows))
* [Mac OSX Install](https://bitbucket.org/sage2/sage2/wiki/Install%20(Mac%20OSX))
* [OpenSUSE Install](https://bitbucket.org/sage2/sage2/wiki/Install%20(openSUSE))
* [Ubuntu Install](https://bitbucket.org/sage2/sage2/wiki/Install%20(Ubuntu))

## 1.3. Configure

* Create a [configuration file](https://bitbucket.org/sage2/sage2/wiki/Configuration) for your display environment
* Save file in /config
* Select your configuration file
  + Option 1: name your configuration file '-cfg.json'  
    (eg. host = thor.evl.uic.edu, config file is 'thor-cfg.json')
  + Option 2: create a file 'config.txt' in   
    Specify the path to your configuration file in 'config.txt'

## 1.4. Run

* Open Terminal / Cmd
  + cd <SAGE2\_directory>
  + node server.js
* Open Google Chrome (to enable screen sharing, go to chrome://flags and enable "Enable screen capture support in getUserMedia()")
  + Table of Contents: http://<host>:<index\_port>
  + Display Client: https://<host>:<port>/?clientID=<ID>
  + Audio Client: https://<host>:<port>/audioManager.html
  + SAGE UI: https://<host>:<port>/sageUI.html
  + SAGE Pointer: https://<host>:<port>/sagePointer.html (Allow pop-ups)
* Create a [one button SAGE2 launcher](https://bitbucket.org/sage2/sage2/wiki/Launch%20(Server%20%26%20Displays)) for the server and displays

## 1.5. Supported File Types

* Images
  + JPEG
  + PNG
  + TIFF
  + BMP
  + PSD
* Videos
  + MP4
* PDFs

## 1.6. Notice

SAGE and SAGE2 are trademarks of the University of Illinois Board of Trustees (SAGE™ and SAGE2™).

# 2. Install for Mac OSX (10.7 - 10.9)

**Last revision:** 01 May 2014

## 2.1. Download

* Download [Node.js](http://nodejs.org/) and install (follow installer instructions)
* Download [homebrew](http://brew.sh/) and install (Terminal command creates full install)

## 2.2. Install Dependencies

* Open Terminal
  + brew install ffmpeg --with-libvpx --with-libvorbis
  + brew install poppler --without-glib
  + brew install imagemagick --with-webp

## 2.3. Clone SAGE2

* Open Terminal
  + cd <directory\_to\_install\_SAGE2>
  + git clone https://bitbucket.org/sage2/sage2.git

## 2.4. Generate HTTPS Keys

* Open the file 'GO-mac' inside the '/keys/' folder
  + Add additional host names for your server in the variable servers (optional)
  + Save file
* Open Terminal
  + cd <SAGE2\_directory>/keys
  + ./GO-mac
    - enter your password when asked (the keys are added into the system)

## 2.5. Install Node.js Modules

* Open Terminal
  + cd <SAGE2\_directory>
  + npm install

# 3. Install for Ubuntu (14.04)

**Last revision:** 01 May 2014

## 3.1. Download

* Download [Node.js](http://nodejs.org/)
* Download [FFMpeg](http://www.ffmpeg.org/download.html) (2.2.1 "Muybridge" gzip tarball)

## 3.2. Install Dependencies

* Open Terminal
  + sudo apt-get install g++
  + sudo apt-get install libx264-dev
  + sudo apt-get install yasm
  + sudo apt-get install imagemagick
  + sudo apt-get install libnss3-tools
  + sudo apt-get install git

## 3.3. Install Node.js

* Open Terminal
  + cd <Downloads\_directory>
  + tar xzvf <downloaded\_nodejs\_tar.gz>
  + cd <extracted\_nodejs\_directory>
  + ./configure
  + make
  + sudo make install

## 3.4. Install FFMpeg

* Open Terminal
  + cd <Downloads\_directory>
  + tar xzvf <downloaded\_ffmpeg\_tar.gz>
  + cd <extracted\_ffmpeg\_directory>
  + ./configure --enable-gpl --enable-libx264
  + make
  + sudo make install

## 3.5. Clone SAGE2

* Open Terminal
  + cd <directory\_to\_install\_SAGE2>
  + git clone https://bitbucket.org/sage2/sage2.git

## 3.6. Generate HTTPS Keys

* Open the file 'GO-linux' inside the '/keys/' folder
  + Add additional host names for your server in the variable servers (optional)
  + Save file
* Open Terminal
  + cd <SAGE2\_directory>/keys
  + ./GO-linux

## 3.7. Install Node.js Modules

* Open Terminal
  + cd <SAGE2\_directory>
  + npm install

# 4. Install for Windows (7 or 8)

**Last revision:** 01 May 2014

## 4.1. Download

* Download [Node.js](http://nodejs.org/) and install (follow installer instructions)
* Download [7-Zip](http://www.7-zip.org/) and install (follow installer instructions)
* Download [ImageMagick](http://www.imagemagick.org/script/binary-releases.php#windows) and install (follow installer instructions)
* Download [FFMpeg](http://ffmpeg.zeranoe.com/builds/)
* Download [Poppler-utils](http://manifestwebdesign.com/2013/01/09/xpdf-and-poppler-utils-on-windows/)
* Download [TortoiseGit](https://code.google.com/p/tortoisegit/wiki/Download) and install (follow installer instructions)
* Download [Git for Windows](https://msysgit.github.io) (follow installer instructions)

## 4.2. Install FFMpeg

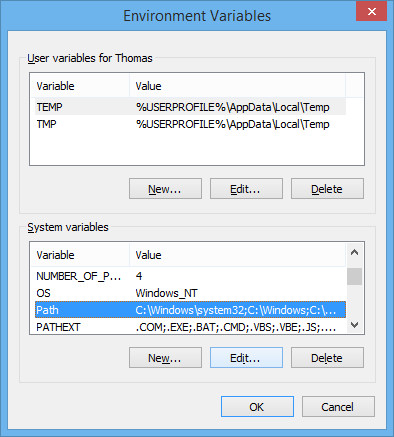
* Move downloaded FFMpeg 7z file to 'C:'
* Right-click on downloaded FFMpeg 7z file
  + 7-Zip > Extract Here
* Rename extracted directory 'FFMpeg'

## 4.3. Install Poppler

* Create directory 'C:'
* Move downloaded Poppler-utils zip file to 'C:'
* Right click on downloaded Poppler-utils zip file
  + 7-Zip > Extract Here

## 4.4. Set Environment

* Add 'C:;C:' to you PATH variable
* You may have to log out and log back in for the environment variables to apply



Edit Windows PATH

## 4.5. Clone SAGE2

In directory where you want to install SAGE2, right-click and select 'Git Clone'

* Set URL to 'https://bitbucket.org/sage2/sage2.git'
* Set directory to where you want to install SAGE2

## 4.6. Generate HTTPS Keys

* Edit '-windows.bat'
  + Add lines with list of hostnames for your server
  + Save file
* Double click 'GO-windows.bat'

## 4.7. Install Node.js Modules

* Launch command prompt (cmd.exe)
  + cd <SAGE2\_directory>
  + npm install

# 5. Install for Linux openSUSE (13.1)

For older versions of openSUSE (and future versions), the name of the packages might change slightly, but the instructions remain mostly valid.

## 5.1. Install Dependencies

* In a Terminal window as 'root' user (or using a sudo command)
* zypper install nodejs : JavaScript environement
* if you prefer a most recent version of NodeJS, you can download the sources from http://nodejs.org/download/ and compile your own:
  + tar xzvf <downloaded\_nodejs\_tar.gz>
  + cd <extracted\_nodejs\_directory>
  + ./configure
  + make -j 5
  + sudo make install
* after install, test:
  + node -v will tell you which version is installed
* zypper install git : distributed revision control system
* zypper install openssl: Secure Sockets and Transport Layer Security
* zipper install mozilla-nss-tools : NSS Security Tool
* zypper install ImageMagick : Viewer and Converter for Images
* zypper install poppler-tools : PDF Rendering Library Tools
* some packages that we use might require the compiler and development packages to be installed
  + zypper install -t pattern devel\_C\_C++
* if you want to use privileged network ports (port 80 for HTTP and 443 for HTTPS), you need to add capabilities to the node binary:
  + zypper install libcap-progs
  + sudo setcap 'cap\_net\_bind\_service=+ep' /usr/bin/node
    - this allows a regular user to use node with privileged ports
* Packages provided by other repositories
  + add repositories:
    - zypper ar http://packman.inode.at/suse/13.1 Packman\_13.1
    - zypper ar http://dl.google.com/linux/chrome/rpm/stable/x86\_64 Google\_chrome
    - zypper refresh
  + zypper install ffmpeg : Viewer and Converter for Images
  + zypper install google-chrome-stable: Google Chrome browser

## 5.2. Clone SAGE2

* Open Terminal
  + cd <directory\_to\_install\_SAGE2>
  + env GIT\_SSL\_NO\_VERIFY=true git clone https://bitbucket.org/sage2/sage2.git
    - enter bitbucket login information when asked

## 5.3. Generate HTTPS Keys

* Open the file 'GO-linux' inside the '/keys/' folder
* Add additional host names for your server in the variable servers (optional)
  + for instance add the short name and the fully qualified domain name of your server
* Save file
* In a Terminal
* cd <SAGE2\_directory>/keys
* ./GO-linux

## 5.4. Install Node.js Modules

* Open Terminal
* cd <SAGE2\_directory>
* npm install

# 6. SAGE2 config

A JSON file to configure the display environment

## 6.1. Fields

|  |  |
| --- | --- |
| Field | Value |
| host | hostname or ip address of the web server |
| public\_host | public hostname of the web server (used in index page) [optional] |
| port | HTTPS port that all clients are served on (default is 443) |
| index\_port | HTTP port where a Table of Contents is served on (default is 80) |
| clock | 12 or 24 (specifies whether to use a 12 or 24 hour clock) |
| background {color, image, style, clip} | color is a hex color for the background, image is a relative path from the publicHTTPS directory to an image used for the background, style is either fit, stretch, or tile, clip is a boolean telling to clip the display at the exact resolution |
| show\_url | boolean (whether or not to show the host url on the display clients) |
| resolution {width, height} | width and height in pixels of a display client (browser window size) |
| layout {rows, columns} | number of rows and columns of display clients that make up the display environment |
| displays [{ID, row, column}] | array of displays with a unique ID (index in array), and the row and column where it tiles in the display environment |
| alternate\_hosts [] | array of alternate hostnames or ip addresses of the web server |
| remote\_sites [{name, host, port}] | array of remote sites to be able to share content with - name to be displayed, host and port specify the remote machine to connect with |
| advanced {ImageMagick} | advanced options not required by all systems: Windows requires full Path to ImageMagick (use / as path delimiter) |

## 6.2. Sample

{  
 "host": "hostname.com",  
 "public\_host": "public.hostname.com",  
 "port": 443,  
 "index\_port": 80,  
 "clock": 12,  
 "background": {  
 "color": "#FFFFFF",  
 "image": "images/background/bgImg.png",  
 "style": "fit",  
 "clip": false  
 },  
 "show\_url": true,  
 "resolution": {  
 "width": 1920,  
 "height": 1080  
 },  
 "layout": {  
 "rows": 1,  
 "columns": 2  
 },  
 "displays": [  
 {  
 "ID": 0,  
 "row": 0,  
 "column": 0  
 },  
 {  
 "ID": 1,  
 "row": 0,  
 "column": 1  
 }  
 ],  
 "alternate\_hosts": [  
 "localhost",  
 "127.0.0.1"  
 ],  
 "remote\_sites": [  
 {  
 "name": "Remote1",  
 "host": "other.com",  
 "port": 443  
 },  
 {  
 "name": "Remote2",  
 "host": "another.com",  
 "port": 9090  
 }  
 ],  
 "advanced": {  
 "ImageMagick": "C:/Program Files/ImageMagick-6.8.9-Q16/"  
 }  
}

## 6.3. Note

Default ports 80 and 443 are convenient when visiting SAGE2 clients because the port does not have to be explicitly entered (eg. in this sample config a user could simply visit https://hostname.com/sagePointer.html instead of https://hostname.com:443/sagePointer.html).

However, these ports require privileged access. If you do not have the necessary permissions, please choose port numbers larger than 1024 and explicitly specify the chosen port numbers when typing a URL.

If a public hostname is specified, it is used on the index page. That host should be listed in alternate\_hosts.

# 7. Starting SAGE2 with one button

**Last revision:** 13 June 2014

## 7.1. Windows

* Single Machine
  + Create file 'GO-.bat' and save in '/GO-scripts'

## 7.2. Mac OSX

* Single Machine
  + Create file 'GO-' and save in '/GO-scripts'

## 7.3. openSUSE

* Single Machine
  + Create file 'GO-' and save in '/GO-scripts'
* Cluster
  + Create file 'GO-' and save in '/GO-scripts'

## 7.4. Ubuntu

* Single Machine
  + Create file 'GO-' and save in '/GO-scripts'

# 8. SAGE2 Application Methods

|  |  |
| --- | --- |
| Method | Parameters |
| init(id, width, height, resrc, date) | id is canvas element id, width is the initial application width, height is the initial application height, resrc is path to resource directory, date is the date |
| load(state, date) | state is the initial application state (eg. initial data to display), date is the date |
| draw(date) | date is the date (used to calculate t and dt for animations). Within your application, call 'refresh' if you need a redraw. If it's interactive application, you can enable 'animation' in the file "instruction.json". the default frame rate is 60fps. Maximum frame rate can be controlled by the variable maxFPS (this.maxFPS = 20.0 for instance) |
| resize(date) | date is the date |
| event(eventType, user\_id, x, y, data, date) | eventType is the type of event, user\_id is the id of the user who triggered the event, x and y are the position of the event, data is an object containing extra data about the event, date is the data . See event description below. |

## 8.1. Boiler-plate for a canvas application

var myApp = SAGE2\_App.extend( {  
 construct: function() {  
 // call the constructor of the base class  
 arguments.callee.superClass.construct.call(this);  
  
 // initialize your variables  
 this.myvalue = 5.0;  
  
 this.resizeEvents = "continuous";//see below for other options  
 },  
  
 init: function(id, width, height, resrc, date) {   
 // call super-class 'init'  
 arguments.callee.superClass.init.call(this, id, <html-tag container for your app (eg. img, canvas)>, width, height, resrc, date);  
  
 // application specific 'init'  
 this.log("Init");  
 },  
  
 //load function allows application to begin with a particular state. Needed for remote site collaboration.   
 load: function(state, date) {  
 //your load code here- state should define the initial/current state of the application  
 },  
  
 draw: function(date) {  
 // application specific 'draw'  
 this.log("Draw");  
  
 //may need to update state here  
 },  
  
 resize: function(date) {  
 // to do: may be a super class resize  
 // or your resize code here  
 this.refresh(date); //redraw after resize  
 },  
  
 event: function(eventType, userId, x, y, data, date) {  
 // see event handling description below  
   
 // may need to update state here  
   
 // may need to redraw   
 this.refresh(date);  
 }  
});

## 8.2. Function Descriptions

### 8.2.1. Construct

### 8.2.2. Init

### 8.2.3. Draw

### 8.2.4. Resize

### 8.2.5. Load

### 8.2.6. Event

#!javascript  
  
event: function(eventType, userId, x, y, data, date) {  
  
}

eventType can be any one of the following:

* pointerPress (button down)
* pointerRelease (button up) -In both of these cases data.button will indicate whether it is 'left' or 'right' button that caused the event.
* pointerMove
* pointerDoubleClick (left button double click)
* keyboard (Normal key events from the keyboard)
* specialKey (Special Keys on the keyboard such as Backspace, Delete, Shift, etc...)

data holds values such as button(for mouse events), state(for key board events to signal 'up' or 'down' state of the key), and code (code of the key pressed in case of key board events).

x and y gives the position of the pointer.

userId gives the Id of the user generating the event.

date object is may be used for synchronizing.

## 8.3. External Libraries

## 8.4. How to write an app

### 8.4.1. instructions.json

### 8.4.2. load into app library

## 8.5. Future: Widgets

## 8.6. Future: Inter-application communication

## 8.7. Example Application: Clock

var clock = SAGE2\_App.extend( {  
 construct: function() {  
 arguments.callee.superClass.construct.call(this);  
 this.ctx = null;  
 this.minDim = null;  
 this.timer = null;  
 this.redraw = null;  
  
 this.resizeEvents = "continuous";  
 },  
  
 init: function(id, width, height, resrc, date) {  
 // call super-class 'init'  
 arguments.callee.superClass.init.call(this, id, "canvas", width, height, resrc, date);  
  
 // application specific 'init'  
 this.ctx = this.element.getContext("2d");  
 this.minDim = Math.min(this.element.width, this.element.height);  
 this.timer = 0.0;  
 this.redraw = true;  
 this.log("Clock created");  
 },  
  
 load: function(state, date) {  
  
 },  
  
 draw: function(date) {  
 // application specific 'draw'  
 // only redraw if more than 1 sec has passed  
 this.timer = this.timer + this.dt;  
 if(this.timer >= 1.0) {  
 this.timer = 0.0;  
 this.redraw = true;  
 }  
  
 if(this.redraw) {  
 // clear canvas   
 this.ctx.clearRect(0,0, this.element.width, this.element.height);  
  
 this.ctx.fillStyle = "rgba(255, 255, 255, 1.0)"  
 this.ctx.fillRect(0,0, this.element.width, this.element.height)  
  
 var radius = 0.95 \* this.minDim / 2;  
 var centerX = this.element.width / 2;  
 var centerY = this.element.height / 2;  
  
 // outside of clock  
 this.ctx.lineWidth = (3.0/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(85, 100, 120, 1.0)";  
 this.ctx.beginPath();  
 this.ctx.arc(centerX, centerY, radius, 0, Math.PI\*2);  
 this.ctx.closePath();  
 this.ctx.stroke();  
  
 // tick marks  
 var theta = 0;  
 var distance = radius \* 0.90; // 90% from the center  
 var x = 0;  
 var y = 0;  
  
 // second dots  
 this.ctx.lineWidth = (0.5/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(20, 50, 120, 1.0)";  
  
 for(var i=0; i<60; i++){  
 // calculate theta  
 theta = theta + (6 \* Math.PI/180);  
 // calculate x,y  
 x = centerX + distance \* Math.cos(theta);  
 y = centerY + distance \* Math.sin(theta);  
  
 this.ctx.beginPath();  
 this.ctx.arc(x, y, (1.0/100.0) \* this.minDim, 0, Math.PI\*2);  
 this.ctx.closePath();  
 this.ctx.stroke();  
 }  
  
 // hour dots  
 this.ctx.lineWidth = (2.5/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(20, 50, 120, 1.0)";  
  
 for(var i=0; i<12; i++){  
 // calculate theta  
 theta = theta + (30 \* Math.PI/180);  
 // calculate x,y  
 x = centerX + distance \* Math.cos(theta);  
 y = centerY + distance \* Math.sin(theta);  
  
 this.ctx.beginPath();  
 this.ctx.arc(x, y, (1.0/100.0) \* this.minDim, 0, Math.PI\*2, true);  
 this.ctx.closePath();  
 this.ctx.stroke();  
 }  
  
 // second hand  
 var handSize = radius \* 0.80; // 80% of the radius  
 var sec = date.getSeconds();  
  
 theta = (6 \* Math.PI / 180);  
 x = centerX + handSize \* Math.cos(sec\*theta - Math.PI/2);  
 y = centerY + handSize \* Math.sin(sec\*theta - Math.PI/2);  
  
 this.ctx.lineWidth = (1.0/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(70, 35, 50, 1.0)";  
 this.ctx.lineCap = "round";  
  
 this.ctx.beginPath();  
 this.ctx.moveTo(x, y);  
 this.ctx.lineTo(centerX, centerY);  
 this.ctx.moveTo(x, y);  
 this.ctx.closePath();  
 this.ctx.stroke();  
  
 // minute hand  
 handSize = radius \* 0.60; // 60% of the radius  
 var min = date.getMinutes() + sec/60;  
  
 theta = (6 \* Math.PI / 180);  
 x = centerX + handSize \* Math.cos(min\*theta - Math.PI/2);  
 y = centerY + handSize \* Math.sin(min\*theta - Math.PI/2);  
  
 this.ctx.lineWidth = (1.5/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(70, 35, 50, 1.0)";  
 this.ctx.lineCap = "round";  
  
 this.ctx.beginPath();  
 this.ctx.moveTo(x, y);  
 this.ctx.lineTo(centerX, centerY);  
 this.ctx.moveTo(x, y);  
 this.ctx.closePath();  
 this.ctx.stroke();  
  
 // hour hand  
 handSize = radius \* 0.40; // 40% of the radius  
 var hour = date.getHours() + min/60;  
  
 theta = (30 \* Math.PI / 180);  
 x = centerX + handSize \* Math.cos(hour \* theta - Math.PI/2);  
 y = centerY + handSize \* Math.sin(hour \* theta - Math.PI/2);  
  
 this.ctx.lineWidth = (2.0/100.0) \* this.minDim;  
 this.ctx.strokeStyle = "rgba(70, 35, 50, 1.0)";  
 this.ctx.lineCap = "round";  
  
 this.ctx.beginPath();  
 this.ctx.moveTo(x, y);  
 this.ctx.lineTo(centerX, centerY);  
 this.ctx.moveTo(x, y);  
 this.ctx.closePath();  
 this.ctx.stroke();  
  
 this.redraw = false;  
 }  
 },  
  
 resize: function(date) {  
 this.minDim = Math.min(this.element.width, this.element.height);  
 this.redraw = true;  
  
 this.refresh(date);  
 },  
  
 event: function(eventType, userId, x, y, data, date) {  
 // this.refresh(date);  
 }  
});