# SAGE2 Documentation

November 17, 2014

SAGE2

1. SAGE2

1.1. Install SAGE2

1.2. Configure SAGE2

1.3. Launch SAGE2

1.4. Develop for SAGE2

1.5. Startup Scripts

1.6. Install for Mac OS X (10.7 - 10.10)

1.7. Install for Ubuntu (14.04)

1.8. Install for Windows (7 or 8.1)

1.9. Install for Linux openSUSE (13.1)

2. SAGE2 config

2.1. Boiler-plate for a canvas application

2.2. Function Descriptions

2.3. External Libraries

2.4. How to write an app

2.5. Widgets

2.6. Future: Inter-application communication

2.7. Example Application: Clock

2.8. Startup using *systemd*

# 1. SAGE2

SAGE2™: Scalable Amplified Group Environment

SAGE™ software enables teams of users to manage their data in a room covered with displays, as if the room were one seamless canvas. SAGE lets users post and manipulate information, visualizations and animations. The goal of SAGE is to make it possible for people to look at large amounts of information and come to conclusions with greater speed, accuracy, comprehensiveness and confidence.

SAGE was originally conceived in 2004. With National Science Foundation (NSF) funding in 2009, the SAGE team was able to continue to develop and harden SAGE, thereby helping over 100 major institutions worldwide better utilize content on their display walls with SAGE.

With NSF funding in 2013, the SAGE team had an opportunity to develop SAGE2, the next-generation SAGE. SAGE is being rewritten from the ground up, using new and emerging technologies.

SAGE2 is browser-based, and everything is done from the Chrome web browser. Special software is no longer needed. By simply dragging and dropping content from one’s laptop to a visual representation of the wall in the browser, one can transfer the content to the wall.

The user interface has been completely redesigned to make it easier and faster for users to work on really large display walls. The SAGE2 interface emphasizes transportable interfaces rather than stationary interfaces so a user can control SAGE2 from anywhere in a room (or even away from the room). Entire laptop screens can also be pushed onto windows in SAGE2, again directly leveraging browser technology. SAGE2 also enables users to connect to collaborating SAGE2 walls and send documents among them by simply dragging and dropping the content in specially marked “collaboration folders”.

Custom applications are easily built in SAGE2. While SAGE is written in C++, SAGE2 is written entirely in Javascript. SAGE2 includes a small software toolkit that enables users to develop applications or modify existing applications to work seamlessly across multiple displays in a synchronized fashion. This unleashes the possibility of creating a new generation of applications that can take advantage of the special qualities of ultra-high-resolution display walls.

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

### 1.0.1. Install SAGE2

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

### 1.0.6. Install for Mac OS X (10.7 - 10.10)

##### 1.0.6.0.1. Download

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

##### 1.0.6.0.2. Install Dependencies

* Open Terminal
  + brew install ffmpeg --with-libvpx --with-libvorbis
  + brew install imagemagick --with-ghostscript --with-webp
  + brew install exiftool

##### 1.0.6.0.3. Clone SAGE2

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

##### 1.0.6.0.4. 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)

##### 1.0.6.0.5. Install Node.js Modules

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

### 1.0.7. Install for Ubuntu (14.04)

##### 1.0.7.0.1. Download

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

##### 1.0.7.0.2. 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
  + sudo apt-get install libimage-exiftool-perl

##### 1.0.7.0.3. Install Node.js

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

##### 1.0.7.0.4. 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

##### 1.0.7.0.5. Clone SAGE2

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

##### 1.0.7.0.6. 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

##### 1.0.7.0.7. Install Node.js Modules

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

### 1.0.8. Install for Windows (7 or 8.1)

##### 1.0.8.0.1. Download

* Download [7-Zip](http://www.7-zip.org/) and install (follow installer instructions)
* Download [Node.js](http://nodejs.org/) and install (follow installer instructions)
* Download [ImageMagick](http://www.imagemagick.org/script/binary-releases.php#windows) and install (follow installer instructions)
* Download [Ghostscript](http://www.ghostscript.com/download/gsdnld.html) and install (follow installer instructions)
* Download [Git for Windows](https://msysgit.github.io) and install (follow installer instructions)
* Download [TortoiseGit](https://code.google.com/p/tortoisegit/wiki/Download) and install (follow installer instructions)
* Download [FFMpeg](http://ffmpeg.zeranoe.com/builds/)
* Download [ExifTool](http://www.sno.phy.queensu.ca/~phil/exiftool/)

##### 1.0.8.0.2. Install FFMpeg

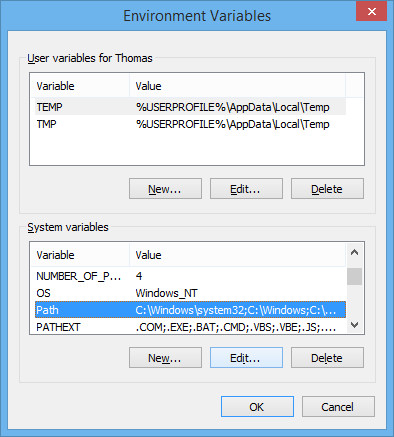
* Create folder 'C:'
* Right-click on downloaded FFMpeg 7z file
  + 7-Zip > Extract Here
* Copy all files from extracted directory to 'C:'

##### 1.0.8.0.3. Install ExifTool

* Right-click on downloaded Exiftool file
  + 7-Zip > Extract Here
* Rename binary to 'exiftool.exe' and move to 'C:'

##### 1.0.8.0.4. Set Environment

* For Windows 7
  + Right-click 'Computer'
  + Click 'Properties'
  + Click 'Advanced system settings'
* For Windows 8.1
  + Click Windows Logo and type 'e'
  + Click 'Edit the system environment variables'
* Click 'Environment Variables'
* Add 'C:;C:Files (x86)' to your system PATH variable
* You may have to log out and log back in for the environment variables to apply



Edit Windows PATH

##### 1.0.8.0.5. 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

##### 1.0.8.0.6. Generate HTTPS Keys

* Edit '-windows.bat'
  + Add lines with list of hostnames for your server
  + Save file
* Double click 'GO-windows.bat'
  + On Windows 8.1 you will need right click and select 'Run as Administrator'

##### 1.0.8.0.7. Install Node.js Modules

* Launch command prompt (cmd.exe)
  + cd <SAGE2\_directory>
  + npm install
  + 'Error: ENOENT, stat 'C:{User Name}'
    - You will need to manually create that folder

### 1.0.9. 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.

##### 1.0.9.0.1. Install Dependencies

* In a Terminal window as 'root' user (or using a sudo command)
* The default version of nodejs provided by default is usually pretty old. It seems better to add the NodeJS repository to the system and install it from there:
  + zypper ar http://download.opensuse.org/repositories/devel:/languages:/nodejs/openSUSE\_13.1/ Node.js
  + zypper in nodejs nodejs-devel npm
* 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 exiftool : Metadata extraction for files
* zypper install poppler-tools : PDF Rendering Library Tools
* zypper install xdotool : Tools to fake mouse/keyboard input and move the mouse outside the viewport
* 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

##### 1.0.9.0.2. 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

##### 1.0.9.0.3. 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

##### 1.0.9.0.4. Install Node.js Modules

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

#### 1.0.1.1. 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'

#### 1.0.1.2. Run

* Open Terminal / Cmd
  + cd <SAGE2\_directory>
  + node server.js (options: -i interactive prompt, -f <file> specify a configuration file)
* Open Web Browser
  + Google Chrome
    - First time use - install [SAGE2 Chrome Extension](https://chrome.google.com/webstore/detail/sage2-screen-capture/mbkfcmpjbkmmdcfocaclghbobhnjfpkk)
  + Firefox
    - First time use - go to about:config and set media.getusermedia.screensharing.enabled to true and add domain(s) for SAGE2 server(s) to media.getusermedia.screensharing.allowed\_domains
  + SAGE2 pages
    - 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

#### 1.0.1.3. Supported File Types

* Images
  + JPEG
  + PNG
  + TIFF
  + BMP
  + PSD
* Videos
  + MP4
  + M4V
  + WEBM
* PDFs

# 2. SAGE2 config

A JSON file to configure the display environment

### 2.0.1. Fields:

{  
 host: // hostname or ip address of the web server  
 port: // HTTPS port that all clients are served  
 on (default: 443)  
 index\_port: // HTTP port, provides a reroute to HTTPS  
 (default: 80)  
 rproxy\_port: // OPTIONAL: port of the HTTPS reverse  
 proxy (only required for reverse proxy setups)  
 rproxy\_index\_port: // OPTIONAL: port of the HTTP reverse proxy  
 (only required for reverse proxy setups)  
 background: {  
 color: // CSS color for the background  
 (hex, rgba(), etc.)  
 image: { // OPTIONAL:   
 url: // relative path from the publicHTTPS  
 directory to an image used for  
 the background  
 style: // either "fit", "stretch", or "tile"  
 }  
 watermark: { // OPTIONAL:   
 svg: // relative path from the publicHTTPS  
 directory to a monochrome SVG image  
 used for the watermark  
 color: // CSS color for the watermark (rgba() recommended)  
 }  
 clip: // OPTIONAL: boolean, whether or not to clip the  
 display at the exact resolution (default: false)  
 }  
 ui: {  
 clock: // 12 or 24 (specifies whether to use a  
 12 or 24 hour clock)  
 show\_url: // boolean, whether or not to show the host  
 url on the display clients  
 show\_version: // boolean, whether or not to show the  
 SAGE2 version number on the display clients  
 menubar: { // OPTIONAL:   
 backgroundColor: // OPTIONAL: CSS color for the background  
 of the menubar (default: "rgba(0,0,0,0.5)")  
 textColor: // OPTIONAL: CSS color for the text of the  
 menubar (default: "rgba(255,255,255,1.0)")  
 remoteConnectedColor: // OPTIONAL: CSS color for remote sites  
 that are connected  
 (default: "rgba(55, 153, 130, 1.0)")  
 remoteDisconnectedColor: // OPTIONAL: CSS color for remote sites  
 that are not connected  
 (default: "rgba(173, 42, 42, 1.0)")  
 }  
 auto\_hide\_ui: // OPTIONAL: boolean, whether or not to  
 autohide wall UI decoration (default: false)  
 auto\_hide\_delay: // OPTIONAL: integer, number of seconds after  
 which to hide the wall UI (default: 30)  
 titleBarHeight: // OPTIONAL: integer, specify window titlebar  
 height in pixels (default: 2.5% of minimum  
 dimension of total wall)  
 titleTextSize: // OPTIONAL: integer, specify text size of ui  
 titles in pixels (default: 1.5% of minimum  
 dimension of total wall)  
 pointerSize: // OPTIONAL: integer, specify pointer size in  
 pixels (default: 8% of minimum dimension  
 of total wall)  
 noDropShadow: // OPTIONAL: boolean, whether or not to disable  
 drop shadows on wall UI decoration  
 (default: false)  
 minWindowWidth: // OPTIONAL: integer, minimum width for  
 application windows in pixels (default:  
 8% of minimum dimension of total wall)  
 minWindowHeight: // OPTIONAL: integer, maximum width for  
 application windows in pixels (default:  
 120% of maximum dimension of total wall)  
 maxWindowWidth: // OPTIONAL: integer, minimum height for  
 application windows in pixels (default:  
 8% of minimum dimension of total wall)  
 maxWindowHeight: // OPTIONAL: integer, maximum height for  
 application windows in pixels (default:  
 120% of maximum dimension of total wall)  
 }  
 resolution: {  
 width: // width in pixels of a display client  
 (browser window width)  
 height: // height in pixels of a display client  
 (browser window height)  
 }  
 layout: {  
 rows: // number of rows of display clients  
 (browser windows) that make up the display  
 wall  
 columns: // number of columns of display clients  
 (browser windows) that make up the display  
 wall  
 }  
 displays: [ // array of displays  
 {  
 row: // the row where this display tiles in the  
 display wall (row origin starts with zero  
 at left)  
 column: // the column where this display tiles in the  
 display wall (column origin starts with  
 zero at the top)  
 },  
 ... // list length should equal rows\*columns  
 ]  
 alternate\_hosts: [ // array of alternate hostnames for machine  
 (i.e. private network IP, localhost, etc.)  
 ...  
 ]  
 remote\_sites: [ // array of remote SAGE2 sites to be able  
 to share content with  
 {  
 name: // name to be displayed on display wall  
 host: // specify the remote machine to connect with  
 (in conjunction with port)  
 port: // specify the remote machine to connect with  
 (in conjunction with host)  
 },  
 ... // list as many remote sites as desired  
 ]  
 dependencies: {  
 ImageMagick: // full path to ImageMagick (use "/" as path  
 delimiter, required for Windows only)  
 FFMpeg: // full path to FFMpeg (use "/" as path  
 delimiter, required for Windows only)  
 }  
 apis: { // OPTIONAL:   
 twitter: { // OPTIONAL: required only for Twitter enabled Apps  
 consumerKey: // Twitter API 'Consumer Key'  
 consumerSecret: // Twitter API 'Consumer Secret'  
 accessToken: // Twitter API 'Access Token'  
 accessSecret: // Twitter API 'Access Token Secret'  
 }  
 }  
}

### 2.0.2. Minimal Sample:

{  
 host: "hostname.com",  
 port: 443,  
 index\_port: 80,  
 background: {  
 color: "#333333"  
 },  
 ui: {  
 clock: 12,  
 show\_version: true,  
 show\_url: true  
 },  
 resolution: {  
 width: 1920,  
 height: 1080  
 },  
 layout: {  
 rows: 1,  
 columns: 1  
 },  
 displays: [  
 {  
 row: 0,  
 column: 0  
 }  
 ],  
 alternate\_hosts: [  
 "127.0.0.1"  
 ],  
 remote\_sites: [  
 ]  
}

### 2.0.3. Expanded Sample

{  
 host: "hostname.com",  
 port: 443,  
 index\_port: 80,  
 background: {  
 color: "#333333",  
 image: {  
 url: "images/background/dbgrid.png",  
 style: "tile"  
 },  
 watermark: {  
 svg: "images/EVL-LAVA.svg",  
 color: "rgba(255, 255, 255, 0.5)"  
 },  
 clip: true  
 },  
 ui: {  
 clock: 12,  
 show\_version: true,  
 show\_url: true,  
 menubar: {   
 backgroundColor: "rgba(24, 30, 79, 0.5)",  
 textColor: "rgba(235, 230, 175, 1.0)",  
 remoteConnectedColor: "rgba(96, 171, 224, 1.0)",  
 remoteDisconnectedColor: "rgba(100, 100, 100, 1.0)"  
 },  
 auto\_hide\_ui: true,  
 auto\_hide\_delay: 15,  
 titleBarHeight: 30,  
 titleTextSize: 18,  
 pointerSize: 90,  
 noDropShadow: true,  
 minWindowWidth: 35,  
 minWindowHeight: 35,  
 maxWindowWidth: 1920,  
 maxWindowHeight: 1920  
 },  
 resolution: {  
 width: 1920,  
 height: 1080  
 },  
 layout: {  
 rows: 1,  
 columns: 1  
 },  
 displays: [  
 {  
 row: 0,  
 column: 0  
 }  
 ],  
 alternate\_hosts: [  
 "host.private.com",  
 "localhost",  
 "127.0.0.1"  
 ],  
 remote\_sites: [  
 {  
 name: "Remote1"  
 host: "other.com"  
 port: 443  
 },  
 {  
 name: "Remote2"  
 host: "another.com"  
 port: 9090  
 },  
 ],  
 dependencies: {  
 ImageMagick: "C:/Program Files/ImageMagick-6.8.9-Q16/",  
 FFMpeg: "C:/local/bin/"  
 },  
 apis: {  
 twitter: {  
 consumerKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
 consumerSecret: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
 accessToken: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
 accessSecret: "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
 }  
 }  
}

##### 2.0.3.0.1. 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.

### 2.0.4. Starting SAGE2 with one button

##### 2.0.4.0.1. Windows

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

##### 2.0.4.0.2. Mac OSX

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

##### 2.0.4.0.3. openSUSE

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

##### 2.0.4.0.4. Ubuntu

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

### 2.0.5. 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. Application can trigger a resize if needed by calling this.sendResize (newWidth, newHeight) |
| *moved* (px,py, wx,wy, date) | application was moved to (px,py) position in wall space (top-left corner), (wx,wy) contains the size of the wall |
| *event* (type, position, user, data, date) | type is the type of event, position contains the x and y positions of the event, user contains data about the user who triggered the event, data is an object containing extra data about the event, date is the date. See event description below. |
| *quit*() | called when an application is closed |

## 2.1. 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(type, position, user, data, date) {  
 // see event handling description below  
   
 // may need to update state here  
   
 // may need to redraw   
 this.refresh(date);  
 },  
  
 moved: function(px, py, wx, wy, date) {  
 // px, py : position in wall coordinate, top-left corner  
 of the window  
 // wx, wy : width and height of the wall  
 // date: when it happened  
 // may need to redraw   
 this.refresh(date);  
 },  
  
 quit: function() {  
 // It's the end  
 this.log("Done");  
 }  
});

## 2.2. Function Descriptions

### 2.2.1. Construct

### 2.2.2. Init

### 2.2.3. Draw

### 2.2.4. Resize

### 2.2.5. Load

### 2.2.6. Event

### 2.2.7. Quit

event: function(type, position, user, data, date) {  
  
}

*type* can be any one of the following:

* *pointerPress* (button down) - data.button will indicate whether it is 'left' or 'right' button that caused the event
* *pointerRelease* (button up) - data.button will indicate whether it is 'left' or 'right' button that caused the event
* *pointerMove*
* *pointerDoubleClick* (left button double click)
* *keyboard* (printable key events from the keyboard) - data.code is the character code, data.character is the printable character's string
* *specialKey* (key events for all keys on the keyboard including Backspace, Delete, Shift, etc...) - data.code is the character code, data.state is 'up' or 'down'

position has x and y to give the position of the event.

user has id, label, and color to describe the user who generated the event

date object may be used for synchronizing.

## 2.3. External Libraries

## 2.4. How to write an app

### 2.4.1. instructions.json

### 2.4.2. Load into application library

## 2.5. Widgets

Four different types of widgets are available for custom applications. Namely:

* Buttons
* Sliders
* Text Input (Single line) and
* Labels

Any application may request SAGE2 to enable widget support for that application. This is done setting the enableControls property to true (this can be done as a part of the construct function).

construct: function() {  
 ...  
 this.enableControls = true;  
}  
  
init: function(id, width, height, resrc, date) {  
 ...  
 this.controls.<addWidget>(paramObject);  
}

### 2.5.1. Button

To add a button to the widget bar, the application may call addButton function on the controls. AddButton function takes one object with two properties, namely type and action. Type specifies the type of the button (appearance and animation of the button) and action is the callback that is executed when the button is clicked.

The type property can take any one value out of: play-pause, play-stop, rewind, fast-forward, prev, and next.

this.controls.addButton({type:"rewind", action:function(appObj, date){  
 // appObj is the instance of the application that this widget  
 is bound to  
   
}});

### 2.5.2. Slider

To add a slider to the widget bar, the application may call addSlider function on the controls. addSlider function takes an object with the following properties:

* *appObj*: The instance of the application that is bound to this slider. Usually 'this' reference is set as the value of appObj.
* *property*: A string representing the property of the application that is tied to the slider. (A change in the value of appObj.property will be reflected in the slider and vice versa)
* *begin*: starting (minimum) value of the property associated with the slider.
* *end*: last (maximum) value of the property associated with the slider.
* *increments*: The step value change that occurs in the property when the slider is moved one unit. (alternatively, parts can be specified in place of increments, where parts represents the number of increments from begin to end)
* *action*: The callback that is executed after the slider has been moved. This callback is not executed when the property value changes from within the app and in turn affects the slider, it is only executed when the property is changed by the slider.

this.controls.addSlider({  
 begin: ,  
 end: ,  
 increments: ,  
 appObj: ,   
 property: ,   
 action:function(appObj, date){  
 ...  
 }  
 });

### 2.5.3. Text Input

To add a slider to the widget bar, the application may call addSlider function on the controls. addSlider function takes an object with two properties, namely width and action. Width specifies the width of the text input widget (in pixels) and action is the callback that is executed when the Enter key is hit, which marks the end of the input.

this.controls.addTextInput({ width: , action: function(appObj, text){  
 // appObj is the instance of the application that this widget is bound to  
 // text data from the widget is sent to this callback.  
}});

### 2.5.4. Label

To add a label to the widget bar, the application may call addLabel function on the controls. addLabel function takes an object with three properties, namely textLength, appObj and property. textLength specifies the maximum length of the string that will be displayed in the label. appObj is the instance of the application that this widget is bound to. property is the property of the appObj whose value will be converted to a string to be displayed in the label.

The changes to the value of the 'property' are immediately reflected in the text displayed in the label.

this.controls.addLabel({textLength:10,appObj:this, property:"pageVal"});

In the above example, the value of *this.pageVal* will be displayed on the label.

## 2.6. Future: Inter-application communication

## 2.7. 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(type, position, user, data, date) {  
 // this.refresh(date);  
 },  
  
 quit: function() {  
 // done  
 }  
});

## 2.8 Startup using *systemd*

If you have SAGE2 installed in a production environment, you may want to add it to your startup scripts so it starts with every reboot.

**Note**: This only starts the node sage2 server, not the display clients!

### 2.8.1. Prerequisites

Your OS distribution needs to be using *systemd* to startup items.

### 2.8.2. Installation

Edit GO-scripts/sage2Server.service and set the WorkingDirectory to your SAGE2 directory.

sudo cp sage2Server.service /usr/lib/systemd/system/ sudo systemctl daemon-reload

### 2.8.3. Enabling sage2Server to start on boot

sudo systemctl daemon-reload

### 2.8.4. Start sage2Server

sudo systemctl start sage2Server

### 2.8.5. Stop sage2Server

sudo systemctl stop sage2Server

### 2.8.6. Restart sage2Server

sudo systemctl restart sage2Server

### 2.8.7. Reload the script

Reminder: Every time you modify the sage2Server.service file, you also need to run:

sudo systemctl daemon-reload