Double Encore: Firebolt Application
Colorado School of Mines
Summer Field Session 2014

Marcus Bermel
Sarah Falkenstein
Paulo Iza
Trevor Westphal

Table of Contents

I. Introduction

i. Application Background and Client Description

This project was originally conceived by Chad Wachs, a member of the Golden Fire Department, and brought to the attention of Eric Daugherty, an employee at the company Double Encore. Together, with the help of three Colorado School of Mines students, they were able to design and complete an application to help the Golden Fire Department better navigate the multitude of surrounding counties' maps.

This year, the FireBolt application team has been tasked with furthering the app and making it run in a scaled-up environment. This generally includes the tasks of enabling multiple departments, and optimizing the app for large-scale usability. Not only did the backend have to be updated, but the client also wished for the user interface to be redone in order to comply with current industry standards.

Over the course of the project, we found ourselves working in close contact with the client, Chad Wachs. He was vital to providing the necessary feedback on design decisions. He was also invaluable as a partial team memeber, constantly bug testing and performing voluntary quality assurance, alerting us whenever a build had the inevitable quirks.

ii. Product Vision

Since its inception, the FireBolt application has been facilitate the ease of navigation regarding the host of city and county maps required to sift through by Emergency Officials. This was furthered this year by the members of the FireBolt app team. We took the original premise and extended it to work in an large-scale environment that far exceeded its original design specifications.

The original product was serviceable, but was not extensible. This was a main requirement; namely, the ability to add in Fire Departments arbitrarily. This was done by reconstructing the backend processes of the application.

This feature, extensibility, is the mainstay of the updates. There were also visual updates to make the app more intuitive to use, as well as secondary functions that made the app more versatile in its functionality.

II. Requirements

i. High-Level Description of Product

Firebolt is an iOS based mapping application that was implemented at the request of the Golden Fire Department to help them respond to emergencies more efficiently. Firebolt currently allows the fire department to upload PDF files to a Dropbox account, which then

pushes to the software on an as needed base. The application allows firemen to zoom into specific grid locations, and scroll from grid location to grid location to access the specific region they need.

ii. Functional Requirements

Top priority functional requirements include producing a scalable product (in order to expand beyond a set number of areas), removing intrusive grid highlighting and text, and implementing swipe navigation when utilizing "metro maps."

iii. Non-functional Requirements

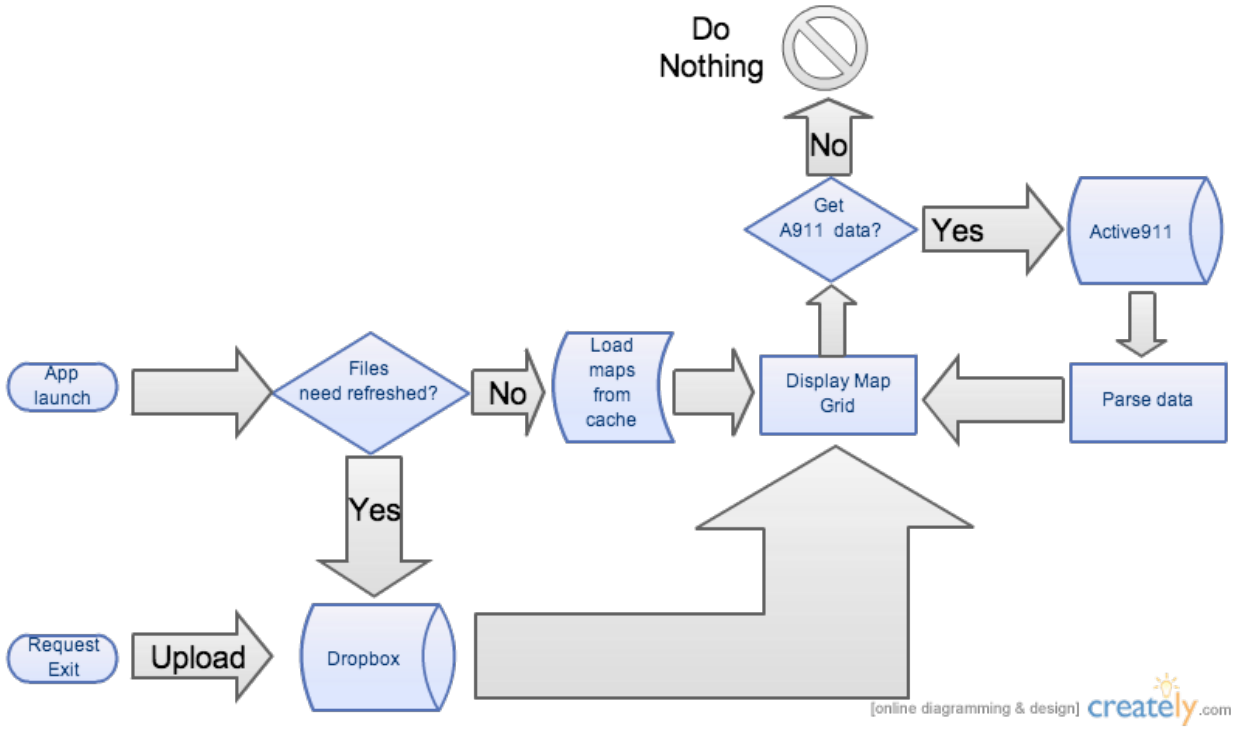
The secondary requirements were to integrate the third-party emergency dispatching software Active911 into the app. Also, the user interface was overhauled to make it iOS 7.1 compatible, as well as generally sleeker looking and more useable. This was done by swapping out text-based buttons with symbolic, intuitive interfaces.

iv. Potential Project Risks

Deployment and full-scale production has inherent risks, being that this is a tool used by emergency response teams. Any errant load times or errors slow down firefighters, who could potentially use that time to be saving lives. This gives the project an air of extreme importance. There also exist skill risks, namely the lack of collective skill in using the technologies used to develop application software. For example, we needed to learn Xcode, the iOS integrated development environment, and Objective C, the primary language used on iOS.

III. System Architecture

The system architecture is comprised of two external data sources: Dropbox and Active911. When the application is launched, data is retrieved from Dropbox and copied into the local cache. Upon exit, any modified data is uploaded to Dropbox so others have access. Active911 services are triggered at the user’s request, so if the user doesn’t request it, nothing happens. If they do, data is retrieved from Active911, parsed, and translated into a map location.



IV. Technical Design

i. Overview

Firebolt is an iOS based mapping application that was implemented at the request of the Golden Fire Department to help them respond to emergencies more efficiently. Our team is currently working to implement Active 911 integration, adding drag and drop clickable links, reorganizing the Dropbox file structure to allow the integration of multiple fire departments, and structuring the layout such that a config.txt file specifies the ordering of PDFs on the grid.

Different from the initial design, Active911 integration now requires user interaction. The user will tap the Active911 button in the navbar, at which point Bolt will pull the latest emergency call information from their API and translate that into a working map location. Map data will be automatically rendered and displayed for the user at this time.

While some gate key symbols already exist on the grid, none of them currently link to the Gate Index PDF file. We want to not only make these symbols link to Gate Index page, but we also want to allow firefighters to add new key symbols on the fly. After dragging and dropping a gate symbol, allow firefighters to set the Gate ID text. Each key symbol should automatically link to the Gate Index PDF. There is also the need for adding a drag-able preplan file symbol, which will be implemented in the same manner as the key symbols, with the exception that rather than asking for a Gate ID text we will ask for a PDF file name (the name of the specific pre-plan).

Although Dropbox has some restraints on the sharing of specific types of folders, we are looking in to allowing fire departments to share their 'Map' folders across Dropbox. Due to security concerns, we cannot simply create a master Dropbox account with global access. Instead, we want to allow each individual fire department to manage their own maps, and also share their maps with other fire departments if they wish.

As seen in Golden's fire department grid map, occasionally the grid 'expands' in unexpected ways, causing the grid names to be ordered in a pattern other than A through Z. While Golden's grid used to be labeled A, B, C, and so forth in descending order, Golden fire department's area of responsibility grew towards the north, making the new grid ordering system BB, AA A, B, C and so on. Our upgrades will allow a config.txt file to specify any letter ordering system that a fire department may use. This change will reduce artificial constraints from the application, and allow more fire departments to use it.

More specifications were added during a later design meeting. These new features and

A few more specifications that were added during our most recent design meeting include:

Create a 'Home' button on each page of app for quick navigation to main map
Update the view for clickable content to use just an outline as opposed to a blue box
Allow user to choose a default department out of list of multiple departments

The overall organization of our code can be seen in our UML document (See Appendix A).

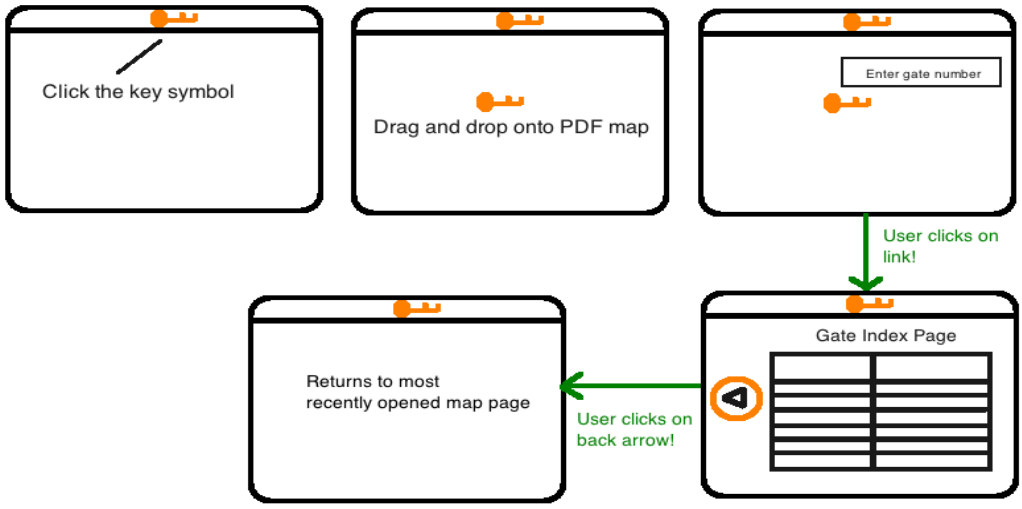
ii. Active 911 Integration Design

Do to lack of push notification support in the Active911 API and iOS limitations regarding reading alerts displayed by other apps, Active911 functionality must be manually triggered. When a user presses the Active911 button in the navbar, the Active911 API is accessed for the latest emergency call data. Once the data is received, it’s translated into a map location, and the corresponding map is displayed.

iii. Adding Drag and Drop Links

A new feature that the Golden fire department has requested is the ability to drag and drop symbols onto their maps. The first symbol, a key symbol, should allow them to link the symbol back to the gate index PDF. The other symbol, used for pre-plans, should allow them to choose a specific PDF filename to link to. While this feature is easily implemented within a MapView, the code is not currently setup to allow us to do this. Instead, we want to be able to add links to our PDFs. After some research, the open source library Haru seemed like a feasible option, but it does not allow us to easily edit PDFs that are already created. Luckily, iOS has a built in library called 'Quartz' that should meet our needs. Also, a similar feature has already been implemented with the blue overlay links (the code for which can be found in the OverlayPDFView class).

1. Click on key symbol in tool bar
2. Click to place key (Drag and drop will be preferable)
3. Enter gate number for link
4. Link should 'save' automatically after created
5. *Return to previous page!



iv. Dropbox Restructuring

Since fire departments may occasionally respond to calls from a nearby area, it would be useful for Firebolt to allow the sharing of map grids. For example, the Golden fire department may want to share their maps with Boulder while still maintaining 'master' control of their own maps. Ideally, each department should have their own dropbox account with the ability to share their folders with other departments. Due to the sensitive nature of some the maps (i.e. gate codes and building plans), a public file option is not desirable.

This was accomplished by simply reconfiguring the Dropbox setup and API calls. First, we had to reconfigure the dropbox account to accept shared files. Then, we had to create a new "App" in dropbox's developer options. We gave this new "App" access to the entire dropbox account, from the root directory. This allowed the "App" to view and modify files that were shared with the current dropbox account.

Finally, we integrated the dropbox "App" with Firebolt. We gave Firebolt the "App" secret code and key pair to correctly communicate with the Dropbox "App". Then, we changed the instantiation of the DropBoxHelper object to reflect the new "Root" level access.

This completed the portion of the actual dropbox restructuring, but did not allow the user to select the department. This was accomplished by integrating a department switching popover, allowing the user to choose the department.

The client then requested that a default department auto-load on the startup of the application. This was done via additional settings menus and a defaultDepartmentController class. This is called once at the installation of the application, and can be modified later in the settings menu.

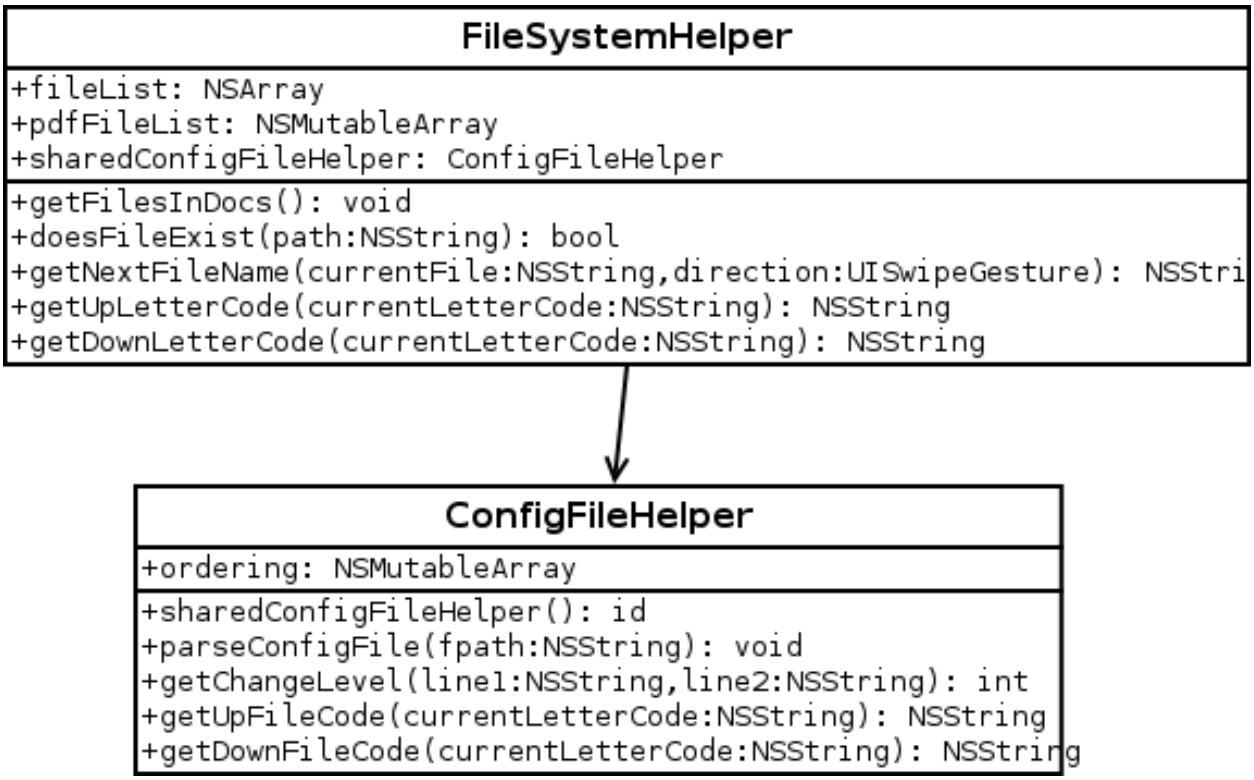
v. Creating Configuration File

To implement this feature, we created a class called ConfigFileHelper that implements the logic for determining the next letter in our pattern, given the config.ord information. This is done by leveraging a custom domain-specific language. Very simply, this language defines the alphabetic order of the maps in a concise and unambiguous manner. The full documentation and specifications of this language and its syntax can be found in Appendix B.

ConfigFileHelper then parses this config.ord file, extracting the order of the maps. This is stored in an array that will be later accessed by FileSystemHelper.

When a new map is loaded, the main thread queries FileSystemHelper for the maps above, below, right, left, and in all diagonal directions from the current map. This is partially dispatched from FileSystemHelper to ConfigFileHelper to query the up and down directions. ConfigFileHelper then iterates through the aforementioned array. If the current maps' letter code is that of any element in the array, uses the next or previous element in the array if appropriate. Otherwise, it increments or decrements the letter code as defined by the parsing.

ConfigFileHelper then returns the up and down letter codes to FileSystemHelper, which determines whether the files generated by the ordering algorithm actually exists in the directory. If it does, it displays the appropriate navigation arrow and links the button to that pdf page.



V. Design and Implementation Decisions

i. Overview

This document covers the technical aspects of the design decisions we made while adding new features to the Double Encore Firebolt application. Many of the design decisions were based off of our working knowledge of Objective-C and iOS, the existing application code, and input from the Double Encore design team.

ii. Active 911 Integration Design

Initially the client wanted to implement a function that would allow him to launch FireBolt from within the Active911 application, thus launching the proper map for the corresponding call. We soon found out that due to a developer's misunderstanding at Active911, this feature would not be possible. Instead, we decided to implement a user-triggered event within FireBolt that would accomplish the same task. There's a button in the navbar of our app for Active911, and when it's pressed, it downloads the latest call data, and opens the correct map. To make this happen, we needed to implement support for Open Authentication 2.0, as that is the standard used by Active911. Since security is important, this was implemented using Google's Toolbox for Mac (GTM).

In order to make everything more convenient for our client, we decided to use self-generated tokens in lieu of application-specific tokens. This allows the client to easily update authentication tokens to get things up and running as quickly as possible. Because of this, the GTM library required a few modifications, as we were no longer following the authorization flow from beginning to end. Methods were added to allow us to insert our manually entered and created credentials into the GTM authentication mechanism, allowing it to function as desired.

All methods pertaining to Active911 are contained within an Active911 class, which allows us to easily access Active911 information from anywhere in the project. This presented a challenge, though, as the data retrieved from the Active911 object had to be passed back to the View Controller somehow, but the call to download data from Active911 is asynchronous, so a standard getter wouldn't work well due to timing complications. To address this, we took advantage of iOS's built in Notification Center. We added a notification observer to the View Controller class, which looked for a notification with a specific tag. Once the asynchronous method completed and it's completion selector is called, a notification matching the description that the observer is looking for is posted, at which point the View Controller can then retrieve the data.

After the data has been copied to the View Controller, we made sure that the currently loaded department was the same as the department from which the call originated, that way when a map grid is loaded using a map code, there's no chance of an error due to nonexistent map grids. If the current department doesn't match, we switch the department, then load the proper grid.

iii. Adding Drag and Drop Links

After extensive research into how to add drag and drop functionality, and how to add links to pre existing PDFs, we decided to implement the least complicated option possible. Overlaying a MapView onto the existing views to get the drag and drop functionality would have required an extensive amount of refactoring, and the open source library Haru would have forced us to create a new PDF file each time we wanted to add a link to it, and then delete the old PDF afterwards.

Instead, after delving further into the PDFView class and ViewController, we decided to implement most of our functionality here. The PDFView class already handles the creation of all of the other views for the application, so we simply created a new UIView class called DragSymbolView to handle our drag and drop functionality. Then within PDFView we add an instance of DragSymbolView as a subview to add our drag and drop symbol to the screen. Within the DragSymbolView class we have four methods called touchesBegan, touchesMoved, touchesEnded, and touchesCancelled, which handle the movement of our symbol. Although we originally wanted to implement a UIPanGestureRecognizer here instead, we ran into problems with the iPad interpreting our 'pan' gestures as swipes, which caused issues within the ViewController.

Given a better understanding of iOS, replacing these four methods with a UIPanGestureRecognizer may be more elegant, but for our current purpose our 'touches' methods works well. We did however, refactor code within ViewController so that the swipes between pages are now handled by UISwipeGestureRecognizers, which seems to be the accepted method of implementing this functionality in iOS, and reduces errors between our 'touches' methods and swiping between pages.

For adding the actual 'link' functionality that we want from our drag and drop symbols, rather than edit the individual PDFs, we decided to use a configuration file to tell us when to draw links on the existing PDF. This reduces many of the complications with editing a PDF on the fly through iOS, and will also allow firemen to delete key symbols and preplans from the map if they so desire. When each new PDF page is loaded, we scan our “links.json” file, and if we find entries matching the name of our current PDF file, then we use the coordinates provided there to draw in the appropriate links. Given the detailed nature of this information, rather than use a .txt file we decided to use a JSON file since a parser for JSON already exists. A sample “links.json” file looks something like this:

```
{
   "gateKeyLinks": [
      [
         "C2.pdf",
         417,
         465
      ]
   ],
   "preplanLinks": [
      [
         "B3.pdf",
         539,
         90,
         "ApartmentPreplan.pdf"
      ]
   ]
}
```

After implementing our key links by using a combination of our DragSymbolView class and DragSymbolFileHelper class, we simply refactored it to be able to also handle our preplan links as well in order to maintain dry code. We created an enum at the top of DragSymbolView that allows us to keep track of whether a view is a KEY or a PRE-PLAN, and then within our code we have checks for the enum type in order to implement the specific, small changes between the two types of view.

iv. Dropbox Restructuring

To meet the requirement of the Client that several people would be able to share documents and have them be persistent across dropbox accounts, we had to restructure the

DropBoxHelper object, as well as the actual database. Mostly, we modified precise lines of code that dictated the protocol of the DropBox access, changing the keys and associated address. In the database, we physically restructured the files.

The files are now on the root level, following a similar pattern. Each department's folder is on the root level, and contains a Maps folder, which in turn must have a grid.pdf, logo.png, and a config.ord file. This will be expanded to include the JSON file mentioned in the drag-and-drop section. The departments folder also contains an Images subdirectory, wherein pre-rendered PNG's are held. This helps the pdf viewer render faster.

v. Creating a Configuration File

Given the requirement that firemen be able to write and understand their own grid configuration files, it was important for us to avoid anything too technical such as XML or JSON for storing the customizable information. The custom *.ord file can easily be opened, read, edited, and saved with a basic editor. Since the FileSystemHelper class already handled the hard-coded process of determining the next letter from A to Z, this is where we decided to create an instance of our new class ConfigFileHelper.

Since we wanted to add several methods for the specific task of creating and understanding our configuration file, it made sense to make a new class rather than simply expand upon FileSystemHelper. It is a singleton class, meant to have only one instance ever. Upon instantiation, it is given a config file and told to parse it. Then, it stores the ordering of the parsed file in an NSMutableArray. This happens once upon instantiation of the program, and then every time the department is switched. Then, when the viewController requests "Which navigation arrows (up/down/right/left) should I draw?", it queries configFileHelper to tell it what comes before 'A' or after 'AC'. ConfigFileHelper then responds appropriately, giving the coordinates of the grids above and below.

VI. Results

i. Active911 Integration

Part of our project was to integrate a third-party real-time emergency call notification service, Active911, into the application. At first, we were told by Active911 that their application supported the ability to simply copy/paste a URI in order to create a button that would launch Firebolt. That turned out to be an unimplemented feature, so instead the design was changed. Now there's a button in the navigation bar that, when pressed, downloads emergency call data and automatically displays the map grid corresponding to the location of the most recent call.

Another design decision was determining how we would choose to authenticate to the Active911 service. The backend utilizes Open Authentication 2.0, so we had the option of getting application specific access tokens, or creating self generated tokens. Due to the fact that only a small handful of firefighters have admin panel access on Active911, the application specific key was ruled out because they don't have credentials to log in and get an authorization

code. Instead the client now generates their own tokens and can share them with other firefighters, who can then paste the tokens into Firebolt and use them.

Lastly, during development of the Active911 class of Firebolt, the Active911 API changed the format of the data it returned, and the process to obtaining the data the app requires. Normally the parser for the data returned by the server would simply be modified to read only the new format, for performance reasons. In this case we were able to leverage the properties of various objects in order to simultaneously support both the old and the new formats, without taking a performance hit. The more important reason behind this decision is the fact that the odds are pretty good that Active911 will revert to the old format since the new process of obtaining data utilizes more SQL database calls on their backend.

ii. Drag and Drop Annotations

Another request from the client was to add a 'drag and drop' functionality to create links to the gate index page and any preplan pages. We were able to add a draggable view that allows the user to set these link locations on the grid pages. One constraint that we ran into at this point was an issue with detecting drag and drop motions versus swipe motions that take you from page to page. However, by setting some booleans we were able to turn off swiping while we are dealing with an 'unsaved' link.

After clicking the key or pre-plan button to obtain a draggable view, the button becomes a save button. The user can then click this button to save their link. We implemented an alert here so that the user knows their data was saved, and we also quickly reload the page so that our updated grid page can immediately be seen without having to navigate away from the page and come back. Any time a grid page is loaded, the application parses the JSON file for the grid names, and if any of the grid names match the name of the current PDF file being displayed, then a link is drawn at the coordinates given in the JSON file.

If users create a bad link by entering a name for a pre-plan that does not exist, whenever the non-functioning link is clicked, an alert pops up informing the user that the file they are looking to link for could not be found in Dropbox, and that they should add the file to Dropbox and try again. The application then deletes the appropriate link data from the JSON file, so that there aren't any 'dead' links being drawn on the maps.

iii. DropBox Sharing

This section of the project was decided by the client to configure the dropbox setup in a way that facilitated the sharing of dropbox directories. Additionally, The need to add support for multiple departments to the design was necessitated by the client. This included a new style of maps that had three layers of grid pages, rather than the standard two layers. Finally, the GUI needed to be redone and brought up to iOS 7.1 standards.

The dropbox configuration itself turned out to be two lines of code changed, but was a very arduous and delicate process. We had to trace each DropBoxDelegate object, up to 167 threads, to an origin DropBoxHelper object creating them. This was an interface with

DBinstanceWithDelegate, which is part of the core dropbox API. This communicated directly with the DropBox servers. After finding the instantiation of this in the "main" function, we discovered it was authenticated with an app key and secret. Additionally, it was initialized with a certain option, "DropBoxAppDirectory".

We then went into a testing dropbox account, created a new "App" connection in the account, and put the app keys and secrets into the line mentioned above. Then, we changed the option on the object creation to "DropBoxRootDirectory", meaning the app has access to all files in the dropbox account. This means it was able to access multiple directories, some of which could be shared from other accounts.

Some of these departments were the aforementioned Metro style of grids. To detect them, a regular expression was set up to find any files that had a metro styled pattern. Then, when following links in a grid view, the app counted the "level" at which you were at, and acted accordingly. The most difficult section was applying the navigation arrows.

Finally, the redoing of the GUI was fairly painless, excepting the difference between iOS 7 and 6. A good rule of thumb is, if it looks good in 6, it won't in 7, and vice-versa. Each OS handles drawing primitives differently, so a rounded rectangle in 7 will be about 20% larger in 6. This just requires going back and forth, tweaking them both to look good. This was not inherently difficult, but time consuming.

iii. Multiple Departments and Default Departments

Since the app was getting recognition among other fire departments, it had to be updated so it is usable for multiple departments, not just Golden. As mentioned in the dropbox sharing section, we needed to change the root of the app in DropBox and see if shareable files would work with the app. This was necessary to access multiple departments via the app.

There is a button that will lead you to a popover view of all the departments that are currently in the DropBox account a user is linked with. This lets the user see his/her default department as a bigger icon in the top layer with smaller icons of each department in a sets of 3 per row at the bottom layer. Clicking any of the icons will take you to the Department's grid.

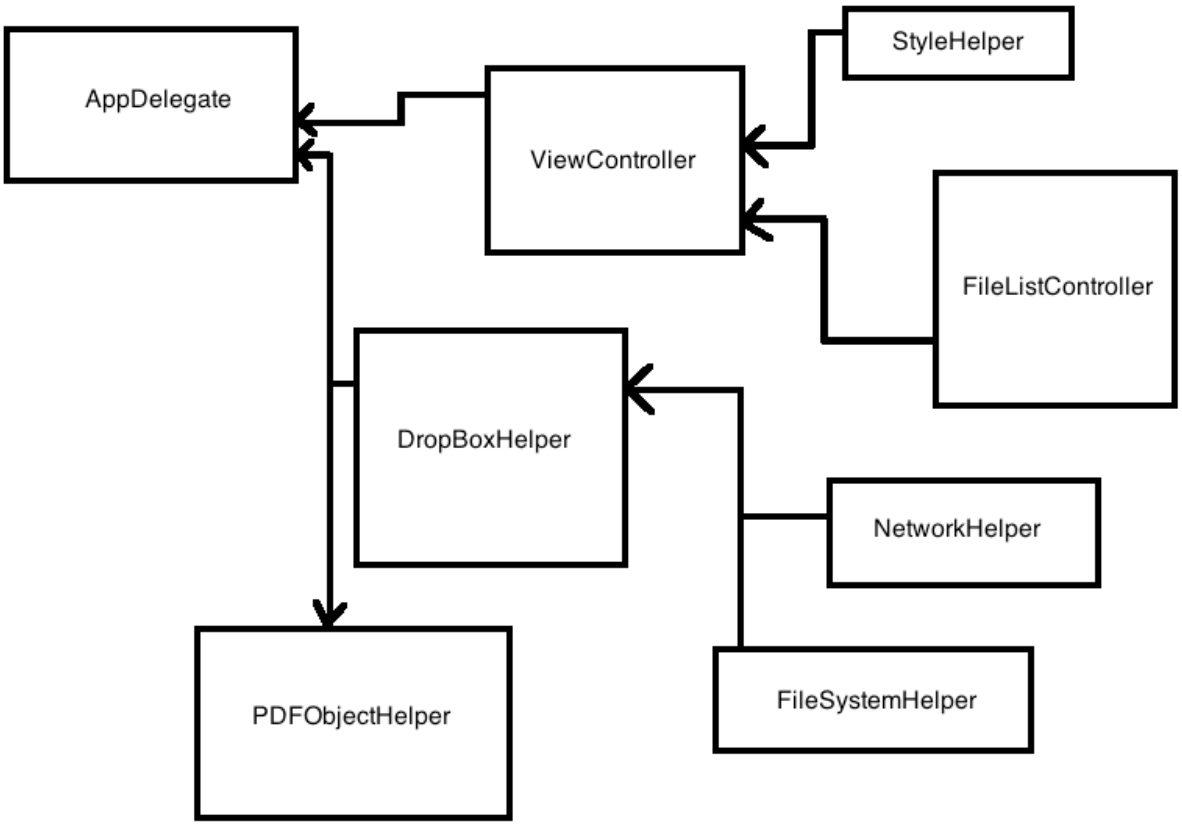
The pop over screen was done with a simple UICollectionView. At first, there were some problems with the nib file, so it ended up being set up programmatically. This different way of making the Collection View helped getting a better understanding of how the nib files do all the work. Now it is easier to deal with nib files.

Selecting a default department is an added feature within the settings button. It will display a list of all the available departments, and clicking on a department will change the default department. This will also be visible on the department button popover with the bigger department icon being the new default department icon.

VII. Appendices

Appendix A: UML Document
Appendix B: Config File Specifications and Syntax
Appendix C: JSON File Instructions
Appendix D: Dropbox Instructions
Appendix E: Firebolt FAQ

Appendix A:


AppDelegate
StyleHelper
ViewController
FileListController
DropBoxHelper
NetworkHelper
PDFObjectHelper
FileSystemHelper

AppDelegate <<UIResponder>>

+window : UIWindow
+viewController : ViewController

-bool didFinishLaunchingWithOptions
-bool handleOpenURL

ViewController

+scrollView : UIScrollView
+navBar : UINavigationBar
+buttons : UIButton
+progressView : UIProgressView
+progressLabel : UILabel
+viewState : viewState
+loadingViewController : LoadingModalViewController

-void viewDidLoad
-void viewDidAppear
-IBAction swipedRight
....
-IBAction downLeftButtonPressed
-IBAction downRightButtonPressed
....
-void presentSettingsViewController
-IBAction homeButtonPressed
-IBAction mapsListButtonPressed
.....

FileListController <<UIViewController>>

+tableViewIsVisible : bool
+currentFileCount : int
+myTable : UITableView
+selectTarget : id
+selectAction : SEL

+void setSelectAction
+NSInteger numberOfRowsInSection

NetworkHelper

+bool hasNetworkConnection

DropBoxHelper

(File paths)
(File flags)

(Downloading functions)
+void clearInstance
+id instanceWithDelegate
.....
+NSString applicationDocumentsDirectory
.....
+void refreshFiles
+void changeDbLinkStatus
+bool isDownloadingFile
+void uploadFile
....

PDFObjectHelper

(PDF flags)
+PDFPage : CGPDFPageRef
+PDFScale : CGFloat
+PDFRect : CGRect
+PDFFileName : NSString
+PDFFilePath : NSString
+PNGFileName : NSString
+PNGFilePath : NSString

(PDF functions)
+PDFObjectHeper getInstance
+void retainPDF
+void releasePDF
.....

FileSystemHelper <<NSObject>>

+fileList : NSArray
+pdfFileList : NSMutable Array

(PDFs management functions)
+void getFilesInDocs
+bool doesFileExist
+appendDocumentsPath : NSString
+appendMapsPath : NSString
.....

Appendix B:

FireBolt Config File Specifications and Syntax

Trevor Westphal
Marcus Bermel
Sarah Falkenstein
Paulo Iza

0. Introduction and Table of Contents

This document is to describe the syntax of the config.ord files used in the iOS Application Firebolt. This descriptor assumes the user knows the basics of the operation of FireBolt, can modify the contents of FireBolt's associated DropBox account, and understands basic programming and computer terminology, including the use of notepad-style text editors.

Code is denoted using the courier font and centered, like this

```
This is code. Code will be about things inside the config
                        file
```

Important things are underlined for clarity

Table of Contents

1. File Placement within the Dropbox File System

There will be one configuration file for each departmental entry in the master dropbox file system. This file will be called “config.ord”, and will be formatted using the UTF-8 standard (The default in most text editors). The file extension, a *.ord, is custom to this application and stands for “Order”. This file can be created and edited using any standard “Notepad” style text editors.

This configuration file will be placed in the following address:

```
/[Department]/Maps/
```

Failure to place the file in the proper directory will result in the file being unable to be read.

2. Basic Syntax

Any line preempted by a double slash will be treated as a comment, and not considered in the ordering of the files. Comments are provided for the end user to annotate any changes or specific notes.

```
// This is a comment, and will not affect the ordering
```

Note that multi-line comments, commonly denoted by /* and */ in some programming languages, will not be considered comments, and will cause errors. Also, note that comments are only considered on a line-by-line basis, so end-of-line comments will not be considered comments.

```
BB // This is NOT a comment, and will cause an error.
```

Any empty line will be ignored

Any line with only uppercase alphabetic characters is a valid entry, and will be considered in the final ordering.

```
AA
```

Entries will be evaluated in pairs, each element of the pair representing the beginning and end of a set. This means there will always be an even number of entries in a config.ord file.

Do not leave any trailing whitespace at the end of lines.

```
AA
```

Note that numbers are not needed to define the orderings of the columns. All columns are assumed to start (at any number) on the left and iterate upwards towards the right. This is done automatically by Firebolt. The config file is concerned only about the vertical ordering of the letters.

3. Simple Ordering

The ordering is derived from the pairs of entries, and the order in which they are in the config file. The file is read from top to bottom, in sets of two. These pairs denote the endpoints of the definition of the set. Say, for example, our ordering went from "A" to "Z", a simple alphabetical ordering. Our config file would consist of only the next two lines (barring any comments).

```
A
Z
```

This tells the computer the ordering begins at "A" and ends at "Z". A more complex, but still valid, ordering could be denoted

```
A
D
Z
E
```

This would be read as such: The first set of definitions is "A" through "D", and the next set is "Z" counting down through "E" (which is supported by this format). If one were to traverse the entire set explicitly, we would see the following ordering:

```
A, B, C, D, Z, Y, X, …, G, F, E
```

The breaks in ordering are not necessary for implicit orderings, like the alphabet. One could write an config.ord file as such:

```
A
G
H
Z
```

Which would iterate through the alphabet, "A" through "G", and then "H" through "Z", but this is not necessary. The format carries with it implicit orderings within a set, including traversing backwards.

Note that one can also skip individual letters or sections. For example,

```
A
D
G
K
```

would skip the letters E and F.

4. Multiple Character Systems

So far, we've only covered single-character systems. Using any number of characters to define the order is possible in this format.

Consider the general double letter format that iterates from "AA" all the way to "ZZ". This can be interpreted two ways, namely

```
AA, BB, CC, DD, …, YY, ZZ
```

or as the much more explicit version,

```
AA, AB, AC, AD, …, AZ, BA, BB, …, ZX, ZY, ZZ
```

This format eliminates this possibility, leaving no ambiguity in the interpretation of the ordering. This has the side effect of having to be more verbose.

Consider the first case, AA, BB, …, ZZ. This can be seen as having a consistent "Level of Change", meaning each entry has all of its characters changed in the same way, namely, incremented. That means, given a character "N", we know we iterate to the next character, "N+1". This would qualify as a set, and can be defined simply as

```
AA
ZZ
```

The second case, AA, AB… AZ, BA, …, ZZ does not have a consistent "Level of Change", meaning we can't apply the incrementation across all the characters. If we were to apply the incrementation across all characters, "AB" would become "BC", which is not correct. This is why this case must be broken up into 26 different subsets, each defined with a consistent "Level of Change". This broken-up definition would give us a final definition:

```
AA
AZ
BA
BZ
CA
CZ
…
XA
XZ
YA
YZ
ZA
ZZ
```

Note that this has been edited with ellipses for brevity. As mentioned before, the explicit definition removes ambiguity at the cost of having to be more verbose. Note that, when using any definitions, you do not need to iterate until the ending of the conventional ordering. For

example, if your maps only go from “AA” to “BE” using the second-case ordering, you only need to have:

```
AA
AZ
BA
BE
```

Note that it is also valid to iterate to the end of the conventional ordering, so it would also work to have:

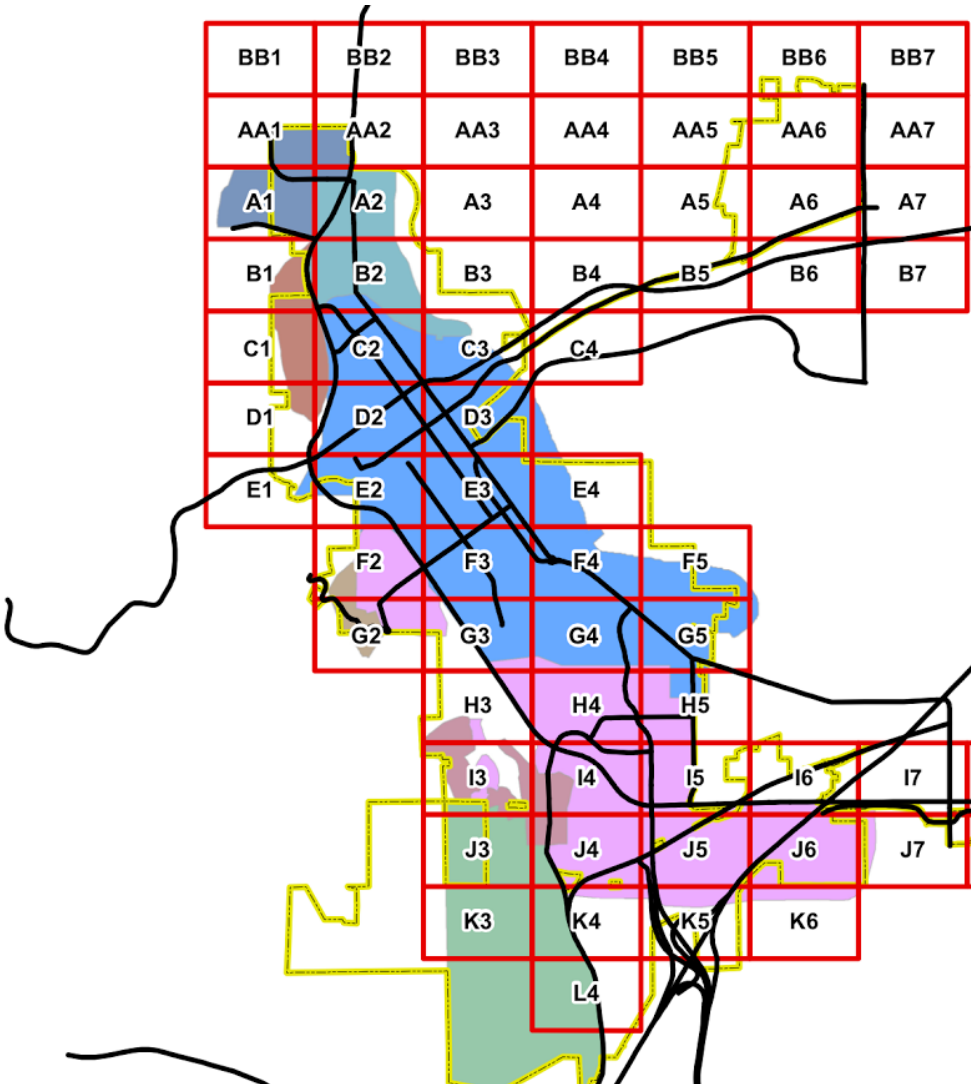
```
AA
AZ
BA
BZ
```

This format may be more helpful than using ‘BE’ at the end of your config file if you know that you may eventually add BF, BG, and so forth underneath BE, and don’t want to have to remember to go in and change the config.ord file.

Keeping these definitions to a minimum helps the app and anyone else who must understand the config files, at the cost of having to update the config file should anything change.

5. Example Config File Creation

We will now walk through the creation of a config file from a source. Here is the city of Goldens map, on the following page.



Note here that the ordering begins with BB and descends to AA. This has a consistent "Level of Change", and should be considered a single subgroup. Next, we can see that it begins lettering at A, then continues to go up to L.

Now, several personal decisions come in, because there are several valid config files possible, and they depend on what changes you would expect from the map in the future. For example, if you expected the city to expand upward, creating CC, DD, etc. zones, you could define the upper bound as such.

If you thought the bottom were to expand, you could iterate past L all the way to Z if you wished.

So, this gives us four possibilities, all of which are valid and would work in the program:

```
// Config.ord for Golden
```

BB AA A L	ZZ AA A L	BB AA A Z	ZZ AA A Z
is most concise, but allows for no expansion. Config must be updated.	Growth upward expected.	Growth downward expected	Growth expected, but not known where. A very safe bet, but possibly not necessary.

I'm going to use the most verbose one. Here is the final config file example.

```
// Config.ord for Golden, written by Trevor Westphal

// Upper bound as "ZZ" for upward expansion
ZZ
AA

// Lower bound as "Z" for downward expansion
A
Z
```

Appendix C:

JSON File Specifications

Trevor Westphal
Marcus Bermel
Sarah Falkenstein
Paulo Iza

The purpose of this document is to detail important user information for the "links.json" file for the iOS application Firebolt.

What is a JSON file and why do I need one?

A JSON file is a special type of file that can easily be read by both computers and humans.

The JSON file stores the information that Bolt needs to draw links to the gate key index page, and to draw links to the individual preplan pages. All you need to do is create an empty file named "links.json", and place it in your department's Maps folder. This file will be a special type of file called a 'JSON' file, which can easily be created using a simple text editor.

Once the empty JSON file has successfully been created and added to the Maps folder, Bolt will automatically add data to it for your gate key links and preplan links.

Editing Existing Gate Key Links and Preplan Links

Link data can be deleted from these files if you later decide that you don't want a link on a given page, but this must be done so very carefully. *If you accidentally created a link to a nonexistent preplan page, simply clicking on this link in the application will give you an alert that the PDF file does not exist, and the application will delete the preplan link data for you- which means you don't need to go through the process of editing the JSON file directly!

To modify or delete link data, start by downloading your JSON file from Dropbox. JSON files are a very special type of file. They are great for storing data that the computer can easily read and understand, but they are also very sensitive to any extra tabs, spaces, and punctuation marks. This is why it is very important to save your backup copy of the JSON file, and to be very careful and precise when editing it. **Make a copy of your JSON file and save it in a safe location on your computer.**

This way if you accidentally corrupt the JSON file while editing it, you will still have a working copy of the file. If you forget to make a safe copy of your JSON file and then corrupt the file, you will have to start over with an empty JSON file and re-add all of your links as you did before (or do some research on JSON files, neither of which will be very fun!).

Our JSON file has two different sections- one section is called "gateKeyLinks" and the other is called "preplanLinks". The order that these two sections appear in your JSON file may vary, but this is okay (i.e. the "preplanLinks" data may actually be displayed above the "gateKeyLinks" data). If we want to delete a gate key link, we will delete data from the "gateKeyLinks" section, and if we want to delete a preplan link, we will delete data from the "preplanLinks" section. On the next page we have a sample JSON file created by Bolt. Our sample file has three gate key links, and two preplan links.

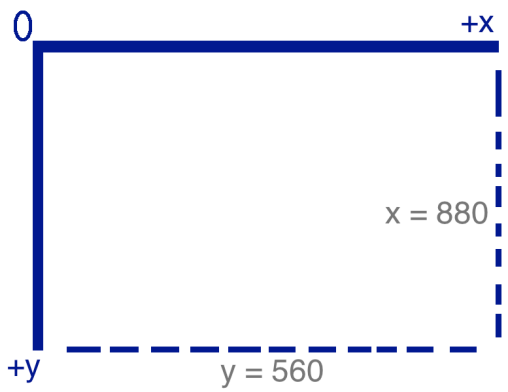
```
{
    "gateKeyLinks": [
        [
            "C2.pdf",
            417,
            465
        ],
        [
            "B4.pdf",
            731,
            313
        ],
        [
            "A2.pdf",
            539,
            90
        ]
    ],
    "preplanLinks": [
        [
            "A3.pdf",
            731,
            313,
            "SchoolPreplan.pdf"
        ],
        [
            "C2.pdf",
            539,
            90,
            "ApartmentPreplan.pdf"
        ]
    ]
}
```

For our example, let's say we have an out of date gate key link that's being drawn on our C2 map page. Open up the JSON file and look under the "gateKeyLinks" section. To delete the key link being drawn on C2, highlight everything from line 3 to line 7 and delete it. Notice that we are deleting all of the text associated with C2, as well as the opening and closing square brackets. Just like an expression such as (3 * 2 + 5 wouldn't make very much sense in math, the computer will be confused if you forget to delete both brackets associated with our data. Our new JSON file can be seen at the top of the next page.

```
{
    "gateKeyLinks": [
        [
            "B4.pdf",
            731,
            313
        ],
        [
            "A2.pdf",
            539,
            90
        ]
    ],
    "preplanLinks": [
        [
            "A3.pdf",
            731,
            313,
            "SchoolPreplan.pdf"
        ],
        [
            "C2.pdf",
            539,
            90,
            "ApartmentPreplan.pdf"
        ]
    ]
}
```

What if I have multiple entries for the same map page and don't know which one to delete?

The easiest way to handle this would probably be to hand draw a quick sketch of the map page the way it is currently rendering, delete all of the JSON entries for the map page, and then go back and re-add the other links that you wanted to keep. If you really want to just delete the one specific entry, then you'll have to compare the coordinates for the link. In the JSON file, following the map page file name is the x coordinate for the link, followed by the y coordinate for the link. Also, make sure to check out an example of Bolt's coordinate system below (with the right-hand bounds of the screen being around 880 pixels, and the lower bounds of the screen being around 560 pixels).



Appendix D:

Creation of a DropBox Account for use with the FireBolt Application

Marcus Bermel
Sarah Falkenstein
Paulo Iza
Trevor Westphal

When creating a DropBox account, several specifications must be met for the app to run.

1. The app must be given access to your DropBox account. This is done in-app, in the settings menu.

2. The app can run in a non-sterile environment, but for best results, include only the necessary maps, files, and directories. This will cut down on load times.

3. When creating a new directory, make sure to follow these instructions:
    1. Create a new folder in DropBox at the root level.
    2. Name this folder the department you wish it to represent. Note, the case and spacing of the letters are sensitive.
    3. Inside this directory, create two folders, called “Images” and “Maps” respectively.
    4. Upload all relevant maps to the “Maps” folder. Be sure to include
        - grid.pdf
        - config.ord
        - links.json
    5. The department is now ready to use. Make sure the links within the PDF files reference local files, and that the links are correct.

4. When sharing folders with other dropbox users, note that the app can and will upload rendered images to your folders. This is normal use for the application, which cuts down on load times for the PDF objects.

5. When another dropbox user shares a folder, it will appear in your account. This means the app will download and use it as if it were a department folder. Only receive shared department folders. Also, be sure to save backup copies of maps and images.

Appendix E:

FireBolt FAQ

Can I put other files in the dropbox directory alongside the departments?

Yes, you can. However, we strongly advise against it, to give the application a sterile, controlled environment in which to operate.

Will shared files from other dropbox accounts appear in Firebolt?

Yes, files that have been shared from other accounts will appear in Firebolt. We discourage the sharing of non-application files, as per above. Note that files shared can and will be modified by Firebolt. For example, the *.json file can be modified from any machine and these changes will reflect across all other devices running Firebolt after a file refresh.

How many departments can I have?

Currently, the app supports up to 9 departments. However, this is discouraged to keep down loading times of the app on installation.

Why is the app taking forever to load on installation?

This is because it is downloading a lot of PDFs. This may take time. We encourage you not to mess with buttons and settings at this time.

Why does the app have slightly misaligned or grey areas in menus?

This is because of iOS 6. Please update your machine to iOS 7. The app is programmatically and visually optimized for iOS 7.

Why is the deparment logo say "Image not Found"?

This means you have failed to place a "logo.png" in your maps directory.

Why are the up and down navigation arrows not appearing?

You have most likely not uploaded a "config.ord" file to dropbox in your maps folder. Please follow the documentation for the config files for further details.

Why are none of the up/down/right/left navigation arrows appearing?

Firebolt was unable to detect maps that correspond to the grid pattern. Make sure your maps increase in number from left to right, and are identified vertically in the "config.ord" file.

Why are the links on the main "grid.pdf" page not working?

Firebolt follows pdf links literally. Make sure the links in your PDF file link to pages in the same directory, like they are in dropbox.

Why is my main page, "grid.pdf" not rendering, but I can still click on links?

+You did not include an empty "/Images" directory in your dropbox folders. Make sure to do this when creating new Department folders.

Why does Firebolt crash when loading any links from grid.pdf?

This occurs when the program is unable to detect a valid "links.json" file. Make sure to upload a valid, empty *.json file as per specification in the documentation.