2011

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

CMobile: A Mobile Photo Capture Application for Construction Imaging

Andrew Martin

A Capstone Project Submitted to the
University of North Carolina Wilmington
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2011

Approved By Advisory Committee

_____
Dr. Ron Vetter, Chair

_____
Dr. Jeff Brown

_____
Dr. Tom Janicki

Abstract

In recent years the mobile application space has exploded in popularity, a fact which is reflected in the drastically increasing availability of both free and paid applications on a variety of platforms. In order to take advantage of this ever-growing market, Construction Imaging has developed a mobile photo capture iPhone application, called CMobile, to supplement their content management solution offerings.

This paper describes the original requirements and features of the application, the methodology by which design choices were tracked and implemented, reviews the issues and problems encountered, the resolutions employed and lessons learned, and concludes with a discussion of the future of the application.

Table of Contents

1. Introduction 4
2. Problem Statement 4
3. Review and Analysis 5
4. Project Methodology, Risks and Related Constraints 8
5. Project and Task Management 13
6. Testing and Verification 16
7. Summary and Conclusions 17
8. Bibliography 18
9. Appendices 19
   - A: Use Cases 19
   - B: Test and Execution Plans 20
   - C: Source Code 32

1. Introduction

Construction Imaging, a leader in industry specific content management solutions, provides a fully comprehensive integration of content workflow, storage, and archiving for construction, engineering, and property management [3].

Over the past decade the company has grown from providing a simple desktop solution to offering a suite of complex, fully customizable products and utilities that foster efficiency, simplicity and reliability on both the desktop and the web.

Construction Imaging has been looking for opportunities to extend these solutions to the exploding arena of mobile applications and devices. This project presents a product geared specifically towards that market.

2. Problem Statement

Recently Construction Imaging has received a request to enhance the content management model to include photograph management. The types of photographs project managers on site tend to take include images of completed or in-progress construction, safety violations or various impediments to job progress.

Currently these photos, once taken, must be extracted from the camera and then uploaded to a computer where they can be manually indexed into Construction Imaging's content management system either via the desktop or web applications. At times the project manager might be out in the field and unable to immediately return to a workstation in order to index the document. If he's using a camera and needs to file his photos immediately, he would have to drop everything he's doing and find a suitable from which location to index the photos. If he's using a web enabled camera-phone there's an easier solution, but still far from ideal: e-mail. Additionally, sending the images to a third-party to index them might be his only option.

But what if there was a way for a user to take photos from his phone and automatically upload them to the system with a set of index values populated with data such as the phone's GPS coordinates, a related job number or other specific, configurable values?

Construction Imaging has seen this need as a great opportunity to develop a solution in the mobile application space, specifically an application designed to take advantage of the features provided by Apple's iPhone 4 [2].

In this paper I present the software development methodology, systems analysis/design, and implementation details of a system called CMobile that allows users to interface with and add

photographic content automatically to Construction Imaging's content management system via his/her iPhone.

3. Review and Analysis

Currently there are no mobile applications on the market that perform the type of photo management anticipated for the CMobile application. A company called Vela Systems provides a similar product titled Vela Mobile that includes an application for the iPad, but only interfaces with the proprietary Vela Field Management Suite [10]. In discussions with Construction Imaging, it was decided to develop a proprietary product that interfaces with and only with Content Manager, thereby giving the company complete control over branding and the ability to fully optimize ease-of-use through seamless integration with existing and potential customers.

The following is the original list of requirements for CMobile.

- Required features
    - CMobile will need to integrate with the phone's camera, allowing the user to take a photograph directly from the application.
    - The photograph must then be processed (compressed if necessary) with index values associated with it from various static and configurable criteria potentially including but not limited to the phone's number, GPS coordinates, an associated job number, and other configurable values, be they predefined for the content type or selectable from a list of keywords.
    - The option to upload the photograph and its accompanying index values via a configurable web service definition to Construction Imaging's content management system must be provided.
    - An option must be provided to allow the photograph and its index values to be emailed directly from the application.
    - The ability to upload photographs that exist on the phone but were not taken from within the mobile application.
    - Access to the application must be restricted via login credentials validated against the web service.
    - Username and password may be saved locally for quick login.
- Optional features
    - Inclusion of the ability to upload/email a batch of photographs with a single set of index values.
    - Inclusion of the ability to attach a voice note/recording as well.

When CMobile launches for the first time it will prompt the user to configure the application's settings (Figure 1). The user must then specify a web service URL address that points to an

exposed Content Manager web service URL. Once the address is verified and the connection is established the user is directed to the Login screen (Figure 2) which will populate the data source, requiring her to enter a username and password, all of which can be saved locally in the settings menu in order to bypass the login screen on subsequent use of the application.

Figure 1: The CMobile Settings Screen ***Figure 2: The CMobile Login Screen***

Once the user has been authenticated he is presented with the Home screen (Figure 3). This screen allows the user to take a photo, index or email a photo or image stored on the phone, or edit the application's settings.

If the user chooses to take a photo he will be presented with the iPhone's camera, otherwise he can browse for a photo or image. Once the user has specified the image he wishes to use he is returned to the home screen where he can choose to index or email the image. If the user opts to index the image he is presented with the index screen (Figure 4).

Figure 3: CMobile's main navigation screen ***Figure 4: Indexing an image***

The user can then enter index values associated with the image. Any fields configured for GPS coordinates or heading are pre-populated from the phone if that data is available. Once the user has completed his input he can upload the image. When the upload has completed the user will be informed of its success and redirected to the Home screen in order to repeat the process again if desired.

The previous process is similar if a user decides to index a photo already stored on the phone, except the user will be presented with the iPhone's camera roll where he can choose the images he'd like to index (Figure 5).

If the user decides he'd like to email the image instead he can follow the same steps in taking or browsing for a photo, but this time he will be presented with the Email screen (Figure 6). Here he can specify recipients, a subject and a message to send along with the attached image.

One thing to note is that once the image is indexed and uploaded to CI Content Manager via the WCF service it is no longer CMobile's concern. It may be saved, placed on a workflow, routed, trigger notifications, etc. CMobile's primary purpose is to create and present the data, not perform any other actions on it.

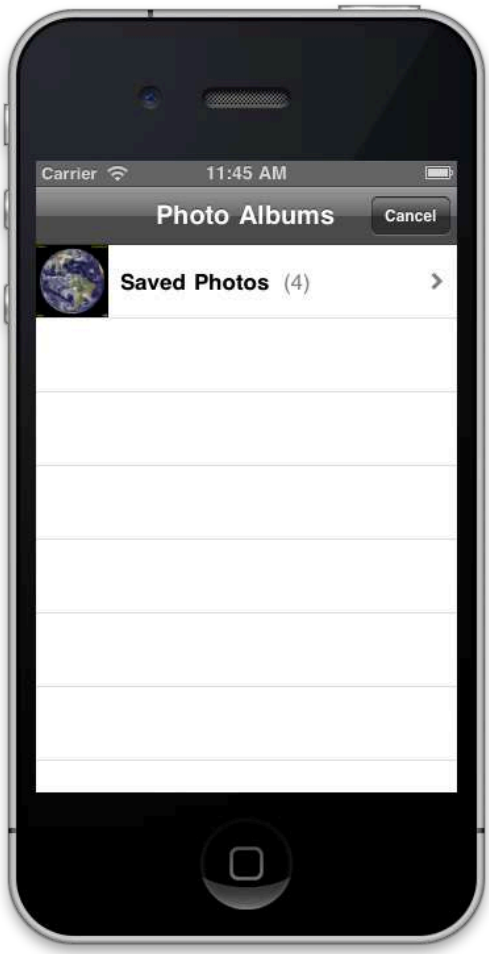


Figure 5: Browsing for an image ***Figure 6: Indexing a photo***

For version 1.0 of CMobile we have decided to not incorporate the ability to upload video or voice recordings, as well as the ability to upload batches of images under one set of index values.

4. Project Methodology, Risks and Related Constraints

In order to produce the mobile photo management application that achieved the objectives of this project, the system was implemented using the following methodologies and technologies:

- The Scrum (Scrum Alliance) agile development methodology [9].

- An Apple Macintosh computer using the iOS development platform provided in the Xcode development IDE (Apple Developer) [1].
- Objective-C using Interface Builder for the user interface development (Apple Developer) [1].
- Communicated with Content Manager (the backend) with a Construction Imaging web service API created using Windows Communication Foundation (WCF) with a JSON enabled endpoint [7].
- All source code was versioned and managed using Microsoft Team Foundation Server via the Team Explorer Everywhere command line tool for OSX [6].
- Development progress was recorded and tracked via Tasks in Microsoft Team Foundation Server's (TFS's) development management tools.
- Functionality was tested by Microsoft Test Manager, which fully integrates will TFS [8].

The Scrum development methodology was chosen over other potential methods due to the fact that it is the primary methodology currently employed by the development team at Construction Imaging Systems. As a small team, we have often encountered the need to quickly adapt to changing requirements and other emerging elements that tend to shift the direction of a development project mid-stream, and have found that a more agile approach, as opposed to the sequential approach of the waterfall model or the somewhat redundant use of an iterative model of development, thus Scrum suits our situation and needs best.

The decision to use Xcode with Objective-C and Interface Builder was reached as said technologies are the standard development tools for Apple's iOS environment. The decision to develop for the iPhone itself instead of other mobile platforms, such as Android, Blackberry, etc., was based on the overall marketability of Apple's product at this time, with potential to expand CMobile to other platforms in the future.

We have previously implemented a web service API using WCF technology that performs some basic tasks for our web products, such as retrieving data source information, authenticating and authorizing user credentials, and importing and saving documents. I used this service to communicate with CI Content Manager.

The following is a list of preexisting service methods that were required for the application:

- **GetDataSources()**
  - A JSON (JavaScript Object Notation) enabled WebGet method that returns a list of available data sources to the user [5].
- **Login(string username, string password)**
  - A JSON enabled WebGet method that authenticates the user based on supplied username and password. Begins the user session.

- **GetLayouts()**
    - A JSON enabled WebGet method that returns the layout of the content type. Includes configuration information about user settings and preferences and available fields and their configurations (name, data type, default value, etc.).
- **Logout()**
    - A JSON enabled WebGet method that ends the current user session.
- **IsLoggedIn()**
    - A JSON enabled WebGet method that determines if the user is still logged and that the session has not expired. Required for instances where the user leaves the application running for an extended period of time. This method is typically called before making any other service calls that require the user session to be active.

The following is a list of previously nonexistent service methods that were required to be added for the application (see Appendix C):

- **GetNewDocument()**
    - A JSON enabled WebGet method that returns a stub of a Document object. In CMobile's case this object contains all the information required to import a photo: its data and index values. This particular method was added so that the JSON representation of the Document object would not have to be constructed programmatically from scratch in the application.
- **GetNewDocumentWithCount(int fieldCount, int fileCount)**
    - A JSON enabled WebGet method that returns a stub of a Document object. It is similar to the GetNewDocument() method but also returns the supplied count of stubs of Fields and Files properties on the Document object.
- **ImportDocumentWithImageStreamAsInvoke(Document document)**
    - A JSON enabled WebInvoke method the document object from the application that describes the photo to be imported into the content management system. This document contains the index field values and the image's data stream.

Figure 7 (below) is a representation of the process by which CMobile communicates with the WCF Service API in order to submit content to the Construction Imaging content management system. The seven service methods that CMobile calls are depicted under the category in which they operate, with the previously nonexistent service methods listed in italics. Note that only the service methods required for the mobile application are listed.

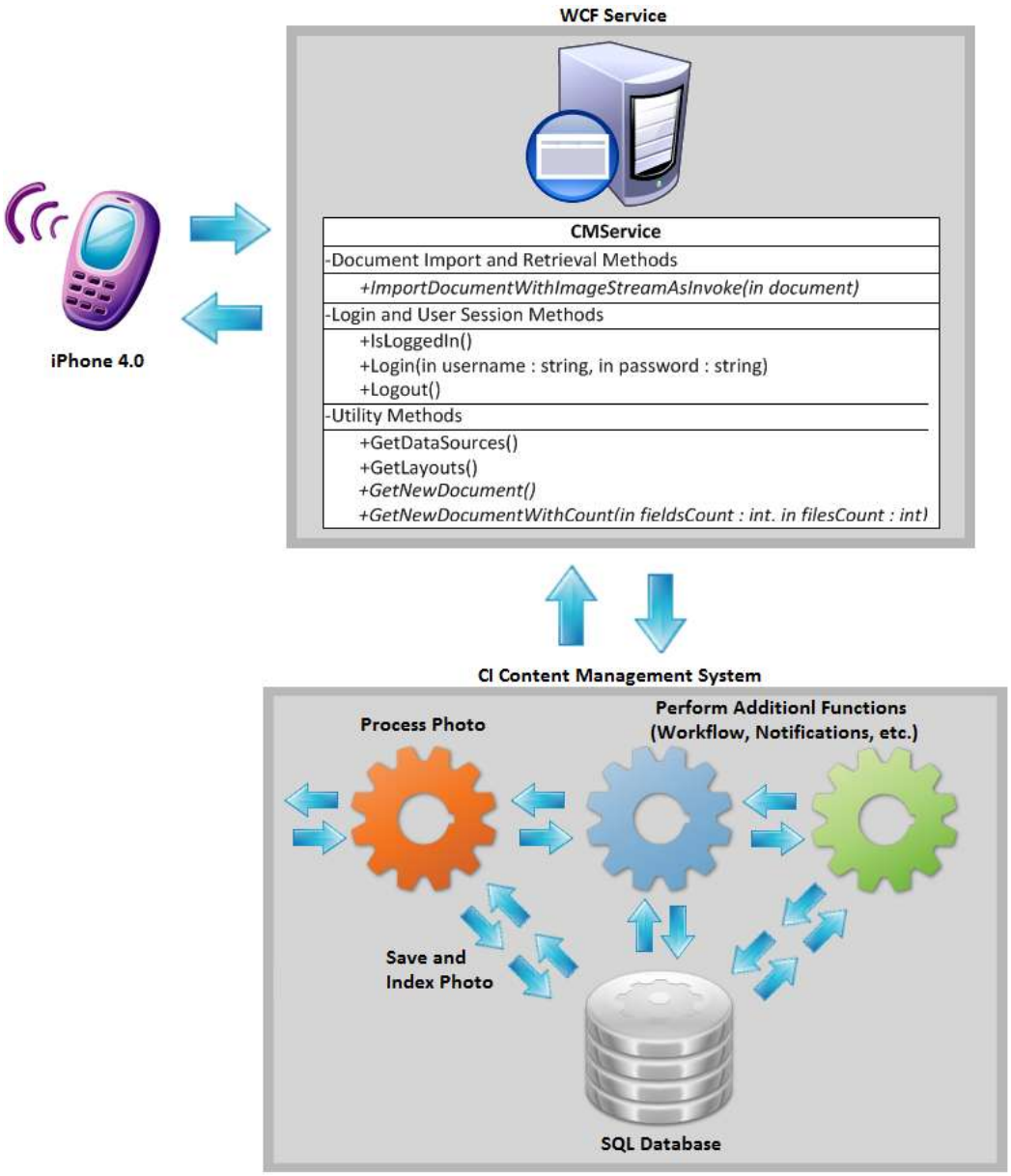


Figure 7: A graphical representation of CMobile and its integration with CI Content Manager

In the original project proposal I predicted that the biggest risk associated with the project would be the iOS development platform itself. Being new to the Apple development environment as well as Objective-C and Interface Builder, I foresaw that perhaps the biggest technological hurdles would arise there, ranging from simple issues such as learning the unfamiliar syntax and properly managing memory, to more complex ones such as integrating with the iPhone's camera. The Objective-C syntax did take some getting used to, and using interface builder in conjunction with Xcode to produce a proper flow between views and their controllers also took a decent amount of time to become proficient with.

Fully understanding memory management also turned out to be somewhat time consuming. The issues included learning when an object needed to be retained in order not to lose reference to it later, when an object needed to be released in order to prevent memory leaks, and when an object should be ignored because another object currently required access to it as well.

Fortunately, Xcode provides both a code analyzer and a real-time leak detection tool. The analyzer can be run on the source code and offers information on locations where memory leaks are bound to occur, areas where leaks may occur, and sections where you might be unnecessarily releasing objects. While the analyzer provides a lot of useful suggestions for handling memory, it can't always account for the flow of the application. For that you need to employ the leak detection tool. You can run the tool alongside your application in the Xcode simulator. It will supply real-time information about current memory allocations, including all introduced leaks. With this tool you can more thoroughly weed out memory issues.

The issue regarding incorporation of the iPhone's camera and email interfaces actually turned out to be the one of the simplest tasks in CMobile's development. Apple has made accessing these features simple, and following a couple of extremely simple tutorials had me integrating with them in short order.

I initially expected the web service communication aspect to present many potential obstacles, as I have had experience communicating via many different protocols in my coursework at UNCW as well as at work, but that figuring out how to consume the service in Xcode might be a unique challenge. In reality, perhaps the most time consuming part of the development process was implementing all the necessary service calls to the CI WCF service. After a few unsuccessful attempts at using various toolkits designed to communicate with non-RESTful services, I discovered that the easiest approach would be to enable certain service methods to return JSON formatted dictionaries and use the iOS JSON Framework to consume/convert these objects. Doing so made accessing data from the service as simple as constructing a RESTful URL and waiting for a response.

Even using the JSON Framework I ran into a few issues trying to pass image data to the service. The first issue was that the service did not accept very large query strings. I updated the size the service would accept and was then able to successfully upload images. The second issue occurred when I realized that images over a certain size (about 1.3 megabytes) would exceed even the maximum allowable query string size. As a result I had to find another method to import the images. The solution was to enable the service methods to accept JSON web invoke calls. This allowed me to configure an HTTP POST message with the image data in the body of the message, bypassing the query string size limits. I was then able to import images of a much larger resolution and quality.

I didn't really see any risk with the Scrum methodology as I've been developing under the system for years, and feel quite comfortable with it at this point. As is common with Scrum a few changes were made to the project along the way, from design, interface and code changes, as well as the decisions to shelve video/audio and batch image imports.

The implementation of the settings menu was completed by using InAppSettingsKit, an open source solution by Edovia [4]. This toolkit allows for the easy inclusion of in-app settings or a duplication of the iPhone's application settings in your application.

All development was focused on creating the application to run on the iPhone, however some of the decisions made along the way, such as the one to use the JSON format for communication, make the project more easily adaptable to other platforms. As discussed, the application communicates with Content Manager via an exposed WCF web service which acts as an API to all the functionality of the content management system itself. Once CMobile has passed its photo and index information to the web service, the content management system is free to perform any number of functions with the photo based on the data associated with it.

5. Project and Task Management

The Gantt chart below described my initial estimate of the time needed to finish the tasks required to complete the application. Each section represented a one to three week iteration or "sprint" as they are referred to in Scrum. These iterations, each containing tasks with their associated work items and change sets, were tracked in Microsoft Team Foundation Server.

ID	Task Name	Start	Finish	Duration
1	Establish communication between application and WCF service.	4/11/2011	4/24/2011	14d
2	Establish application integration with iPhone camera.	4/25/2011	5/8/2011	14d
3	Implement photo index interface	5/9/2011	5/15/2011	7d
4	Implement application settings configuration interface	5/16/2011	5/22/2011	7d
5	Implement login screen interface	5/23/2011	5/29/2011	7d
6	Implement central dashboard/home interface	5/30/2011	6/5/2011	7d
7	Implement photo email functionality	6/6/2011	6/19/2011	14d
8	Test and debug all functionality	6/20/2011	7/3/2011	14d
9	Write defense.	7/4/2011	7/17/2011	14d

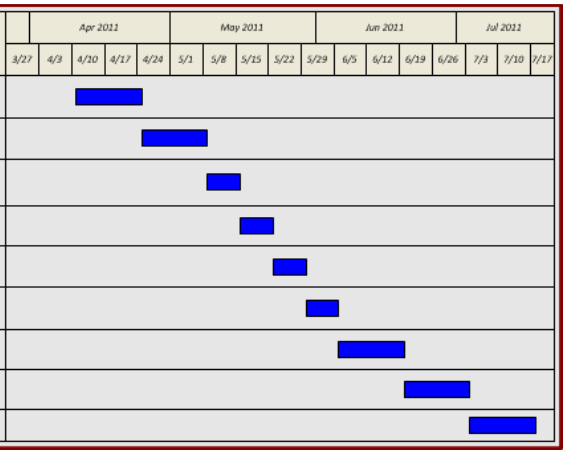


Figure 8: Gantt chart outlining a rough timeline for project completion

Using the requirements and Gantt timeline as a guide I had created a list of basic stories and associated tasks in Team Foundation Server as seen in Figure 9.

Each User Story work item is a rough description of a story or a portion of functionality that needs to be implemented in order to fulfill the requirements of this project. Each story has child tasks that specify work that has to be completed in order for the story to be considered complete. The tasks are the actual items with which all work will be associated, such as code check-ins, change sets, test builds, etc.

The Iteration Path specifies the time period in which each story and its tasks are expected to be completed, with each iteration usually ranging from 1 to 3 weeks. Any tasks not assigned to the current iteration are stored in the iteration backlog until the current iteration is complete, at which time the next iteration is planned, and the next set of tasks are moved from the backlog to that iteration.

Which tasks are moved into the current iteration depends on their Stack Rank, a measure of each story's importance. Stack Rank can be determined by various criteria, but in this particular case the tasks are ranked based on level of importance to the overall development hierarchy in that stories with lower rankings are dependent upon those stories and tasks ranked above them.

ID	Work Ite...	Title	State	Iteration Path	Stack Rank	Story Points	Original Estimate
4482	User Story	Communication between application and WCF service.	Active	Mobile\Iteration 1	1	8	
4483	Task	Implement message passing between WCF service from mobile application	Active	Mobile\Iteration 1			20
4484	Task	Implement the ability to upload an image from the application to the service	Active	Mobile\Iteration 1			15
4498	Task	Create Service Communication Test Suite	Active	Mobile\Iteration 1			3
4485	User Story	Application integration with iPhone camera	Active	Mobile\Backlog	2	5	
4486	Task	Implement iPhone camera integration	Active	Mobile\Backlog			10
4487	Task	Implement iPhone camera roll integration	Active	Mobile\Backlog			8
4499	Task	Create iPhone camera integration test suite	Active	Mobile\Backlog			2
4488	User Story	Photo index and upload	Active	Mobile\Backlog	3	5	
4489	Task	Implement photo index interface	Active	Mobile\Backlog			15
4500	Task	Create photo index test suite	Active	Mobile\Backlog			5
4490	User Story	Application settings configuration	Active	Mobile\Backlog	4	3	
4491	Task	Implement application settings configuration interface	Active	Mobile\Backlog			10
4501	Task	Create application settings configuration test suite	Active	Mobile\Backlog			3
4492	User Story	User Login	Active	Mobile\Backlog	5	3	
4493	Task	Implement login screen interface	Active	Mobile\Backlog			12
4502	Task	Create login interface test suite	Active	Mobile\Backlog			2.5
4496	User Story	User Home Screen	Active	Mobile\Backlog	6	3	
4497	Task	Implement central dashboard/home interface	Active	Mobile\Backlog			8
4503	Task	Create dashboard/home interface test suite	Active	Mobile\Backlog			1.5
4494	User Story	Photo Email	Active	Mobile\Backlog	7	5	
4495	Task	Implement photo email capability	Active	Mobile\Backlog			15
4504	Task	Create photo email test suite.	Active	Mobile\Backlog			2
4505	Test Case	Test manual login - no existing configuration	Ready	Mobile\Backlog			

Figure 9: Team Foundation story and task list as implementation of the Gantt chart described in Figure 7

Story Points are a subjective measurement of how difficult a story will be to implement, based on a factor of risk and estimated time to completion. Each story's value is relative only to the other stories in its specific project.

And finally the Original Estimate is simply an estimation of the time (in hours) that it will take to complete each task.

Figure 10 is a representation of the project in Team Foundation Server upon the completion of all necessary user stories and tasks, as well as the successful execution of the test scenarios associated with each work item.

During development existing items were moved from the project backlog into the next iteration as they were scheduled for work. A couple of bugs not specifically related to tasks for the mobile application were included (cross-referenced) in the mobile project because they hindered the completion of local tasks. Test cases were added to assess functionality of each piece of the application and are discussed in-depth in the next section. Finally, the Completed Work field was updated to represent actual man-hours spent on each portion of the project.

ID	Work Ite...	Title	State	Iteration Path	Stack Rank	Story Points	Original Estimate	Completed...
4482	User Story	Communication between application and WCF service.	Closed	Mobile\Iteration 1	1	8		
4483	Task	Implement message passing between WCF service from mobile application	Closed	Mobile\Iteration 1			20	10.5
4484	Task	Implement the ability to upload an image from the application to the service	Closed	Mobile\Iteration 1			15	17.5
4498	Task	Create Service Communication Test Suite	Closed	Mobile\Iteration 5			3	0
4485	User Story	Application integration with iPhone camera	Closed	Mobile\Iteration 2	2	5		
4486	Task	Implement iPhone camera integration	Closed	Mobile\Iteration 2			10	1.5
4487	Task	Implement iPhone camera roll integration	Closed	Mobile\Iteration 2			8	1.5
4499	Task	Create iPhone camera integration test suite	Closed	Mobile\Iteration 5			2	0.75
4488	User Story	Photo index and upload	Closed	Mobile\Iteration 3	3	5		
4489	Task	Implement photo index interface	Closed	Mobile\Iteration 3			15	13.5
4500	Task	Create photo index test suite	Closed	Mobile\Iteration 5			5	2
4724	Task	Implemented backend service methods	Closed	Mobile\Iteration 3			4	2.5
4490	User Story	Application settings configuration	Closed	Mobile\Iteration 2	4	3		
4491	Task	Implement application settings configuration interface	Closed	Mobile\Iteration 2			10	6.5
4501	Task	Create application settings configuration test suite	Closed	Mobile\Iteration 5			3	1.5
4492	User Story	User Login	Closed	Mobile\Iteration 3	5	3		
4493	Task	Implement login screen interface	Closed	Mobile\Iteration 3			12	8
4502	Task	Create login interface test suite	Closed	Mobile\Iteration 5			2.5	1.5
4496	User Story	User Home Screen	Closed	Mobile\Iteration 4	6	3		
4497	Task	Implement central dashboard/home interface	Closed	Mobile\Iteration 4			8	10.5
4503	Task	Create dashboard/home interface test suite	Closed	Mobile\Iteration 5			1.5	0
4494	User Story	Photo Email	Closed	Mobile\Iteration 4	7	5		
4495	Task	Implement photo email capability	Closed	Mobile\Iteration 4			15	2
4504	Task	Create photo email test suite.	Closed	Mobile\Iteration 5			2	1
4505	Test Case	Test manual login - no existing configuration	Closed	Mobile\Iteration 5				
5097	Test Case	Test automatic login - no existing configuration	Closed	Mobile\Iteration 5				
5098	Test Case	Test manual login failure - no existing configuration	Closed	Mobile\Iteration 5				
5099	Test Case	Test automatic login failure - no existing configuration	Closed	Mobile\Iteration 5				
5100	Test Case	Test manual login	Closed	Mobile\Iteration 5				
5101	Test Case	Test automatic login failure - no existing configuration	Closed	Mobile\Iteration 5				
5102	Test Case	Test Email from Camera	Closed	Mobile\Iteration 5				
5103	Test Case	Test Email from Camera Roll	Closed	Mobile\Iteration 5				
5104	Test Case	Test Index from Camera	Closed	Mobile\Iteration 5				
5105	Test Case	Test Index from Camera Roll	Closed	Mobile\Iteration 5				
5106	Test Case	Test settings changes.	Closed	Mobile\Iteration 5				
5107	Test Case	Test application of image quality setting on Index.	Closed	Mobile\Iteration 5				
5108	Test Case	Test Index from Camera - Session expired.	Closed	Mobile\Iteration 5				
5109	Test Case	Test manual login - Cannot connect to web service URL	Closed	Mobile\Iteration 5				
5006	Bug	Uploading large images to the web service returns a 400 Bad Request error	Resolved	Mobile\Iteration 3				
5007	Bug	None of the index fields for an imported document are being set.	Resolved	Mobile\Iteration 3				

Figure 10: Team Foundation story and task list as implementation upon completion of all work and tests

6. Testing and Verification

In order to verify that the requirements for the completion of this project were met I created test suites in Microsoft Test Manager. Each test suite is associated with a given story or task and has associated test cases that test the entire range of functionality implemented by the story's tasks.

Figure 11 displays a test designed to assess CMobile's ability to index a photo taken from the iPhone's camera, from login to success. Each test is comprised of multiple steps, each with their own expected outcome. As the test progresses, each step can be marked as having passed or failed. The test itself only passes if all of its steps have passed.

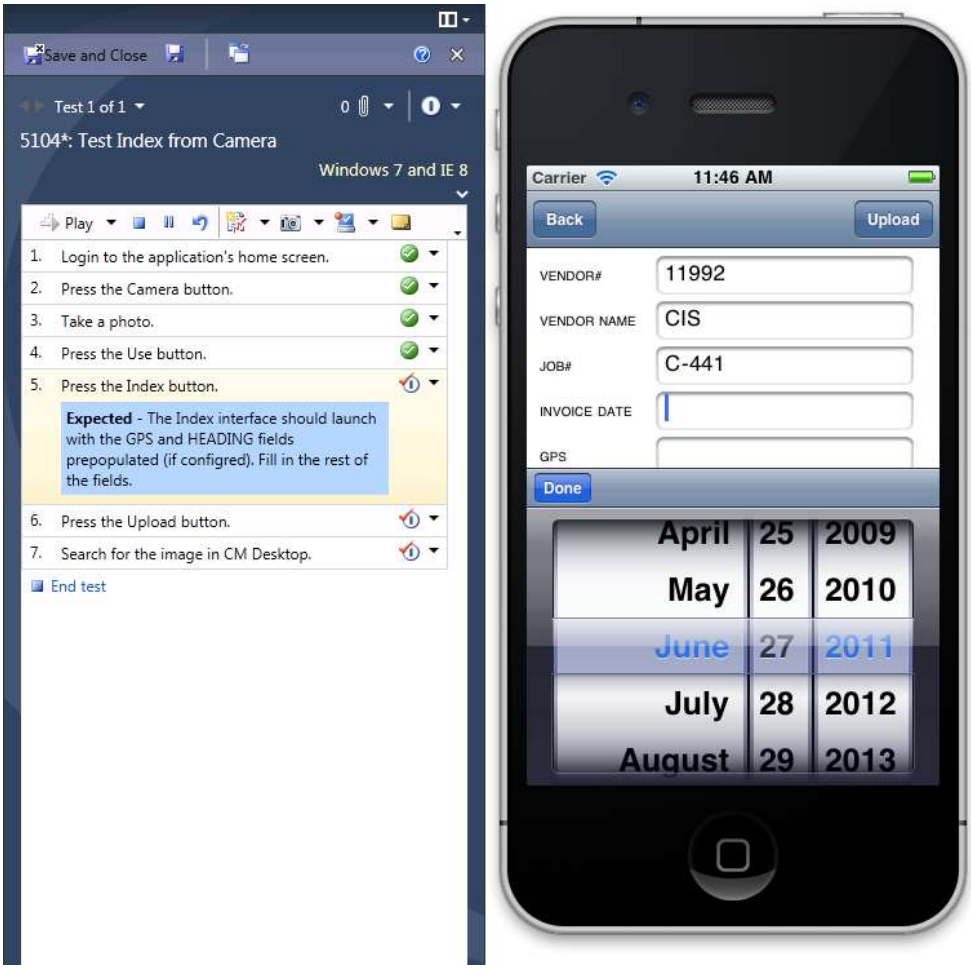


Figure 11: An execution of Test Case 5104 as described in Figure 7

Once every test for every story has passed then the project itself will be complete, having met its requirements. The test and execution plans can be viewed in Appendix B.

7. Summary and Conclusions

CMobile was conceived and commissioned, after reviewing requests for enhancement from both current and potential customers, in order to establish Construction Imaging as an enterprise content management provider in the mobile space, allowing us to expand our suite of products to what most consider the platform of the future in both enterprise content management and computing as a whole.

To further this goal I was charged with creating a simple yet powerful mobile photo capture application for the iPhone. To complete the task I needed to leverage the power of unfamiliar platforms and technologies and integrate them with more recognizable tools and systems in a completely seamless and unified fashion while maintaining both extensibility and adaptability on both sides.

While both Construction Imaging and I are confident that we accomplished what we set out to do in CMobile Version 1.0, the product is far from finished. Ideas for additional features are abundant including, but not limited, to the following:

- Reintroduction of batch image processing.
- The ability to view images and documents in the user's work list and take action on them (approve, reject, review, annotate, attach notes, etc.).
- The ability to view a mapping of all images containing GPS and heading index data in a given area.

This is just a small list of many possible opportunities to expand this project in the future. This first iteration of CMobile will provide a solid platform from which to move forward and realize the full potential of Construction Imaging the mobile space.

8. Bibliography

1. Apple Developer. Xcode - Developer Tools Technology Overview. 2011. 20 February 2011 <http://developer.apple.com/technologies/tools/xcode.html>.

2. Apple Inc. Apple - iPhone 4 - Video calls, multitasking, HD video, and more. 2011. 9 July 2011 <http://www.apple.com/iphone/>.

3. Construction Imaging. Construction Imaging Content Management Software & Systems Integration. 2011. 9 July 2011 <http://www.construction-imaging.com/>.

4. Edovia Inc. Edovia - Insanely Great Software for Mac and iOS. 2010. 9 July 2011 <http://www.edovia.com/>.

5. JSON. n.d. 9 July 2011 <http://www.json.org/>.

6. Microsoft. Team Foundation Server. 2011. 9 July 2011 <http://msdn.microsoft.com/en-us/library/ms181238(v=vs.80).aspx>.

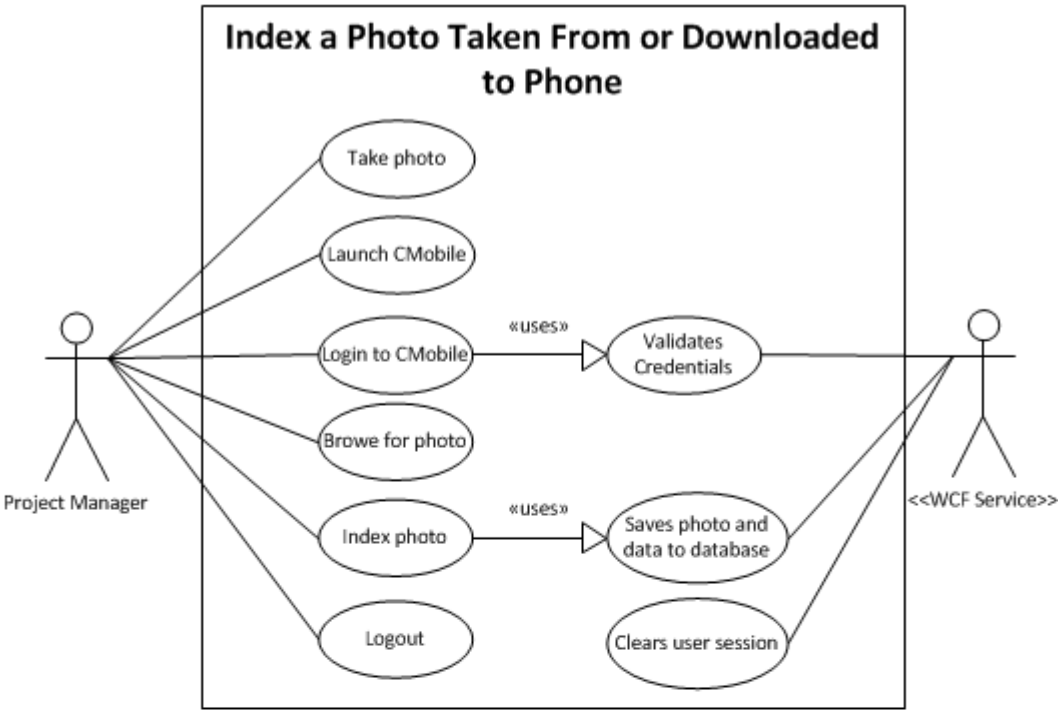
7. Microsoft. What Is Windows Communication Foundation. 2011. 9 July 2011 <http://msdn.microsoft.com/en-us/library/ms731082.aspx>.

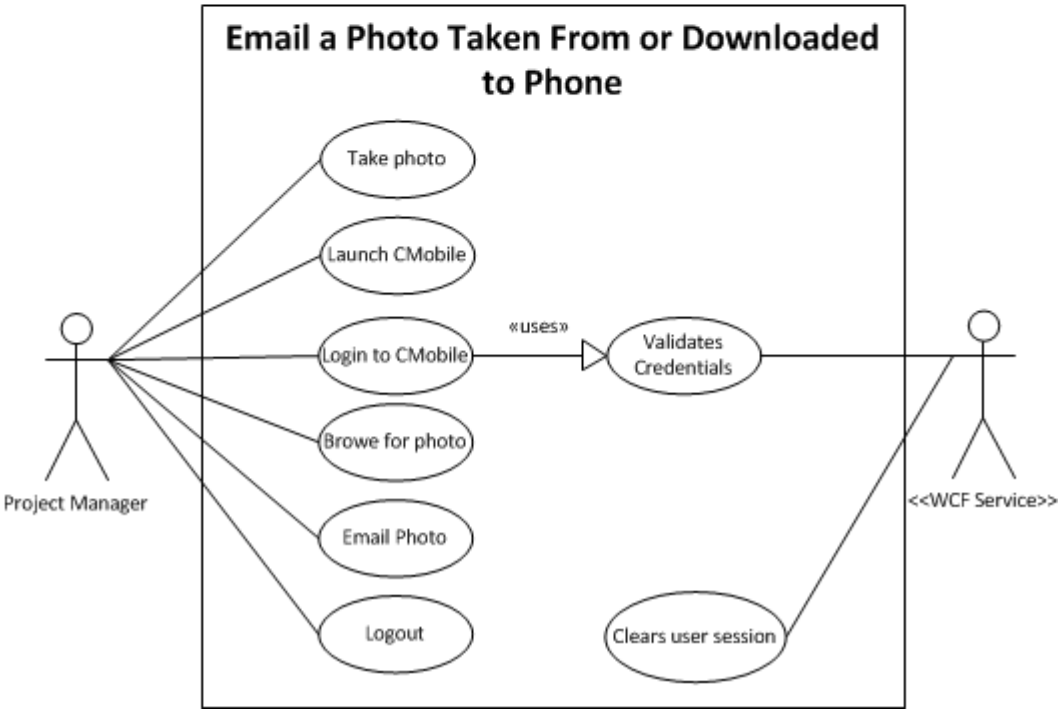
8. Microsoft. What's New For Testing. 2011. 9 July 2011 <http://msdn.microsoft.com/en-us/library/bb385901.aspx>.

9. Scrum Alliance. What is Scrum? 2011. 20 February 2011 <http://www.scrumalliance.org/learn_about_scrum>.

10. Vela Systems. Fela Mobile iPad and Email | iPad App for Construction Field Management. 2011. 03 March 2011 <http://www.velasystems.com/ipad-for-construction-vela-mobile/>.

Appendix A: Use Cases





Appendix B: Test and Execution Plans

Test Plan Details

A test plan lets you specify what you want to test and how to run those tests. A test plan can be applied to a specific iteration of your project. You can have just one default test suite for your test cases, or you can create a test suite hierarchy.

You can also select the default configurations to use to run the tests in your test plan. The test configuration informs the tester of the set up that is required for these tests. You can have one or multiple default configurations. A test result is recorded every time that you run the test with a specific configuration. A test plan enables you to measure your testing progress, based on these test results for the test and configuration pairings.

Test plan 21: CMobile

Status: Active | Active dates: 3/22/2011 12:00:00 AM - 3/29/2011 12:00:00 AM | Area: Mobile | Iteration: Mobile | Build: <Not assigned> | Build definition: <Not assigned> | Build quality: <Not assigned>

Available Configurations (1)

Config 16: iOS 4 for iPhone

Test Settings

Suite Hierarchy

CMobile (0)

- 4488: Photo index and upload (4)
- 4490: Settings configuration (1)
- 4492: User Login (7)
- 4494: Photo Email (2)

Suite Details

You can group your test cases together by organizing test cases into a test suite hierarchy in your test plan. By grouping your test cases together, when you want to run all these tests, you can select this test suite.

Suite 102: CMobile

State: In progress

Available Configurations (1)

Config 16: iOS 4 for iPhone

Test Cases (14)

Test Case 4505: Test manual login - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (4)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Press Done Button.	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source.
	4	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Test Case 5097: Test automatic login - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Type in user name and password.	
	4	Select automatic login option.	
	5	Select remember login information option.	
	6	Close Settings Screen	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source. Username and password should also be populated from the user settings. The screen should attempt to automatically perform user login. Login should complete without error and navigate to the application's home screen.

Test Case 5098: Test manual login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (5)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Press Done Button.	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source.
	4	Type in an incorrect username and password combination. Press Login.	Login should fail and display an appropriate error message to the user.
	5	Type in correct username and password. Press Login.	Login should complete without error and direct to the application's home screen.

Test Case 5099: Test automatic login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	

	3	Type in user name and password.	
	4	Select auotmatic login option.	
	5	Select remember login information option.	
	6	Close Settings Screen	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source. Username and password should also be populated from the user settings. The screen should attempt to automatically perform user login. Login should fail informing the user of the error.
	7	Type in correct user name and password.	Login should complete without error and direct the user to the application's home screen.

Test Case 5100: Test manual login

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (2)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to login screen.
	2	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Test Case 5101: Test automatic login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (1)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen. Login screen should attempt to automatically login and redirect the user to the home screen.

Test Case 5109: Test manual login - Cannot connect to web service URL

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (9)

	#	Title	Expected Value
	1	Launch settings menu from iPhone settings.	Settings should launch
	2	Type in an invalid web service URL and close settings.	
	3	Launch CMobile.	Application should launch to login screen. It should eventually timeout and display a popup alert reading "Cannot load data sources. The request timed out."
	4	Press the Retry button.	The request should eventually timeout and display a popup alert reading "Cannot load data sources. The request timed out."
	5	Minimize the application and return to the CMobile settings menu.	
	6	Type in a valid web service URL and close settings.	
	7	Launch CMobile.	Application should launch to login screen with the popup message still in place.
	8	Press the Retry button	The data source should now populate properly.
	9	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Test Case 5104: Test Index from Camera

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.
	5	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	6	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	7	Search for the image in CM Desktop.	The image should be in the system with the properly associated index values.

Test Case 5105: Test Index from Camera Roll

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Browse button.	The iPhone's camera roll interface should launch.
	3	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	4	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	5	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	6	Search for the image in CM Desktop.	The image should be in the system with the properly associated index values.

Test Case 5107: Test application of image quality setting on Index.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (15)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Click the settings tab.	The settings tab should appear.
	3	Adjust the image quality slider to 100%	
	4	Press the Camera tab.	The Camera tab should appear.
	5	Press the Browse button.	The iPhone's camera roll interface should launch.
	6	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	7	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	8	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a

			popup aleart informing you of the upload's success.
	9	Click the settings tab.	The settings tab should appear.
	10	Adjust the image quality slider to ~50%	
	11	Press the Camera tab.	The Camera tab should appear.
	12	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	13	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	14	Search for the first image in CM desktop.	The image should be in the system with the properly associated index values withest the highest resolution.
	15	Search for the second image in CM desktop.	The image should be in the system with the properly associated index values with an approximately 50% smaller file size (due to increased compression) than the first image.

Test Case 5108: Test Index from Camera - Session expired.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.
	5	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	6	Manually end the session on the server or let the session expire before continuing.	
	7	Press the Upload button.	A popup message should appear reading "Your session has expired. Redirecting to login screen." You should be redirected to the login screen where you will be prompted to login again.

Test Case 5102: Test Email from Camera

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.
	5	Press the Email button.	The iPhone's email interface should launch with a generic subject and body with the image attached. Supply a recipient.
	6	Press the Send button.	You should be redirected to the application's home screen and provided with a popup message informing of the email's success.
	7	Loging into the recipient's email account.	The email should arrive with the proper subject and body with the image attached.

Test Case 5103: Test Email from Camera Roll

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Browse button.	The iPhone's camera roll interface should launch.
	3	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	4	Press the Email button.	The iPhone's email interface should launch with a generic subject and body with the image attached. Supply a recipient.
	5	Press the Send button.	You should be redirected to the application's home screen and provided with a popup message informing of the email's success.
	6	Loging into the recipient's email account.	The email should arrive with the proper subject and body with the image attached.

Test Case 5106: Test settings changes.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (9)

	#	Title	Expected Value
	1	Lauch the application settings screen.	
	2	Change the web service URL setting.	
	3	Change the username setting	
	4	Change the passwrod setting.	
	5	Toggle the remember login information option.	
	6	Toggle the automatic login option.	
	7	Adjust the upload quality slider	
	8	Close the application settings screen	
	9	Open the application settings screen.	All the options should reflect the changes made previously.

Suite 121: 4488: Photo index and upload

State: In progress

Test Cases (4)

Test Case 5104: Test Index from Camera

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.

	5	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	6	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	7	Search for the image in CM Desktop.	The image should be in the system with the properly associated index values.

Test Case 5105: Test Index from Camera Roll

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Browse button.	The iPhone's camera roll interface should launch.
	3	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	4	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	5	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	6	Search for the image in CM Desktop.	The image should be in the system with the properly associated index values.

Test Case 5107: Test application of image quality setting on Index.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (15)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Click the settings tab.	The settings tab should appear.
	3	Adjust the image quality slider to 100%	
	4	Press the Camera tab.	The Camera tab should appear.
	5	Press the Browse button.	The iPhone's camera roll interface should launch.
	6	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	7	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	8	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	9	Click the settings tab.	The settings tab should appear.
	10	Adjust the image quality slider to ~50%	
	11	Press the Camera tab.	The Camera tab should appear.
	12	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	13	Press the Upload button.	A progress bar should appear. When progress completes you should return to the home screen and receive a popup aleart informing you of the upload's success.
	14	Search for the first image in CM desktop.	The image should be in the system with the properly

			associated index values withest the highest resolution.
	15	Search for the second image in CM desktop.	The image should be in the system with the properly associated index values with an approximately 50% smaller file size (due to increased compression) than the first image.

Test Case 5108: Test Index from Camera - Session expired.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.
	5	Press the Index button.	The Index interface should launch with the GPS and HEADING fields prepopulated (if configred). Fill in the rest of the fields.
	6	Manually end the session on the server or let the session expire before continuing.	
	7	Press the Upload button.	A popup message should appear reading "Your session has expired. Redirecting to login screen." You should be redirected to the login screen where you will be prompted to login again.

Suite 124: 4490: Settings configuration

State: In progress

Test Cases (1)

Test Case 5106: Test settings changes.

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (9)

	#	Title	Expected Value
	1	Lauch the application settings screen.	
	2	Change the web service URL setting.	
	3	Change the username setting	
	4	Change the passwrod setting.	
	5	Toggle the remember login information option.	
	6	Toggle the automatic login option.	
	7	Adjust the upload quality slider	
	8	Close the application settings screen	
	9	Open the application settings screen.	All the options should reflect the changes made previously.

Suite 103: 4492: User Login

State: In progress

Test Cases (7)

Test Case 4505: Test manual login - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (4)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Press Done Button.	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source.
	4	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Test Case 5097: Test automatic login - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Type in user name and password.	
	4	Select automatic login option.	
	5	Select remember login information option.	
	6	Close Settings Screen	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source. Username and password should also be populated from the user settings. The screen should attempt to automatically perform user login. Login should complete without error and navigate to the application's home screen.

Test Case 5098: Test manual login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (5)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Press Done Button.	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source.
	4	Type in an incorrect username and password combination. Press Login.	Login should fail and display an appropriate error message to the user.
	5	Type in correct username and password. Press Login.	Login should complete without error and direct to the application's home screen.

Test Case 5099: Test automatic login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen.
	2	Configure Web Service URL to point to a valid service address.	
	3	Type in user name and password.	
	4	Select auotmatic login option.	
	5	Select remember login information option.	
	6	Close Settings Screen	Settings should close and application should navigate to login screen. Data source list should be populated with service's default data source. Username and password should also be populated from the user settings. The screen should attempt to automatically perform user login. Login should fail informing the user of the error.
	7	Type in correct user name and password.	Login should complete without error and direct the user to the application's home screen.

Test Case 5100: Test manual login

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (2)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to login screen.
	2	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Test Case 5101: Test automatic login failure - no existing configuration

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (1)

	#	Title	Expected Value
	1	Launch application.	Application should navigate to settings screen. Login screen should attempt to automatically login and redirect the user to the home screen.

Test Case 5109: Test manual login - Cannot connect to web service URL

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (9)

	#	Title	Expected Value
	1	Launch settings menu from iPhone settings.	Settings should launch
	2	Type in an invalid web service URL and close settings.	
	3	Launch CMobile.	Application should launch to login screen. It should eventually timeout and display a popup alert reading "Cannot load data sources. The request timed out."
	4	Press the Retry button.	The request should eventually timeout and display a popup alert reading "Cannot load data sources. The request timed out."
	5	Minimize the application and return to the CMobile settings menu.	
	6	Type in a valid web service URL and close settings.	
	7	Launch CMobile.	Application should launch to login screen with the popup message still in place.

	8	Press the Retry button	The data source should now populate properly.
	9	Type in user name and password. Press Login.	Login should complete without error and navigate the the application's home screen.

Suite 122: 4494: Photo Email

State: In progress

Test Cases (2)

Test Case 5102: Test Email from Camera

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (7)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Camera button.	The iPhone's camera interface should launch.
	3	Take a photo.	The option to use or discard the photo should be presented.
	4	Press the Use button.	The camera interface should close and return you to the home screen where the photo is now displayed.
	5	Press the Email button.	The iPhone's email interface should launch with a generic subject and body with the image attached. Supply a recipient.
	6	Press the Send button.	You should be redirected to the application's home screen and provided with a popup message informing of the email's success.
	7	Loging into the recipient's email account.	The email should arrive with the proper subject and body with the image attached.

Test Case 5103: Test Email from Camera Roll

Owner: Andrew Martin | Closed | Type: Manual | Mobile | Iteration: Mobile\Iteration 5 | Automated test: Not set | Assigned to: Andrew Martin

Test Steps (6)

	#	Title	Expected Value
	1	Login to the application's home screen.	
	2	Press the Browse button.	The iPhone's camera roll interface should launch.
	3	Select a photo from the roll.	The camera roll interface should close and return you to the home screen where the photo is now displayed.
	4	Press the Email button.	The iPhone's email interface should launch with a generic subject and body with the image attached. Supply a recipient.
	5	Press the Send button.	You should be redirected to the application's home screen and provided with a popup message informing of the email's success.
	6	Loging into the recipient's email account.	The email should arrive with the proper subject and body with the image attached.

Configuration Details

You can use one or more configuration variables to create a test configuration. Each configuration variable defines one characteristic of the testing environment. For example, a characteristic might be the operating system that you want to use to run your tests and the value might be "Windows XP." This test configuration can represent an entry in your test matrix that you plan to use to run tests.

Config 16: iOS 4 for iPhone

15 Test runs passed successfully.

Run by: Andrew Martin | Date started: 7/4/2011 8:34:37 AM | Timespan: 1 day, 18 minutes | Build(s): <Not assigned>

Included Runs (15)

Test Run 433: 4488: Photo index and upload (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 8:34:37 AM | Duration: 56 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 434: 4490: Settings configuration (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 8:35:48 AM | Duration: 13 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 435: 4492: User Login (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 8:36:05 AM | Duration: 38 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 436: 4494: Photo Email (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 8:36:49 AM | Duration: 17 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 441: 4488: Photo index and upload (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:16:19 AM | Duration: 44 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 442: 4490: Settings configuration (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:17:23 AM | Duration: 6 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Local Test Run | Test environment: <Local> | Test controller: <No agent used>

Test Run 443: 4492: User Login (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:17:44 AM | Duration: 38 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 444: 4494: Photo Email (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:18:30 AM | Duration: 20 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 445: 4492: User Login (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:47:14 AM | Duration: 12 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 446: 4488: Photo index and upload (Manual)
Run by: Andrew Martin | Date started: 7/4/2011 9:47:36 AM | Duration: 8 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 449: 4488: Photo index and upload (Manual)
Run by: Andrew Martin | Date started: 7/5/2011 8:50:16 AM | Duration: 38 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 450: 4490: Settings configuration (Manual)
Run by: Andrew Martin | Date started: 7/5/2011 8:51:04 AM | Duration: 9 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 451: 4492: User Login (Manual)
Run by: Andrew Martin | Date started: 7/5/2011 8:51:20 AM | Duration: 46 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 452: 4494: Photo Email (Manual)
Run by: Andrew Martin | Date started: 7/5/2011 8:52:12 AM | Duration: 10 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Test Run 453: 4494: Photo Email (Manual)
Run by: Andrew Martin | Date started: 7/5/2011 8:52:26 AM | Duration: 11 seconds | Run type: Manual | Build(s): <Not assigned> | Test settings: Default test settings | Test environment: <Local> | Test controller: <No agent used>

Outcome Summary

38 Passed (100%)

Unanalyzed Failures (0)

Appendix C: Source Code

```
//
//  CameraViewController.h
//  CMobile
//
//  Created by Andrew Martin on 5/11/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <MessageUI/MessageUI.h>
#import <MessageUI/MFMailComposeViewController.h>


@interface CameraViewController : UIViewController
<MFMailComposeViewControllerDelegate> {
      UIImageView * theimageView;
      UIButton * choosePhoto;
      UIButton * takePhoto;
      UIBarButtonItem *browseItem;
      UIBarButtonItem *cameraItem;
      UIBarButtonItem *uploadItem;
      UIBarButtonItem *emailItem;
      UISlider *qualitySlider;
}

@property (nonatomic, retain) IBOutlet UIImageView * theimageView;
@property (nonatomic, retain) IBOutlet UIButton * choosePhoto;
@property (nonatomic, retain) IBOutlet UIButton * takePhoto;
@property (nonatomic, retain) IBOutlet UIBarButtonItem * browseItem;
@property (nonatomic, retain) IBOutlet UIBarButtonItem * cameraItem;
@property (nonatomic, retain) IBOutlet UIBarButtonItem * uploadItem;
@property (nonatomic, retain) IBOutlet UIBarButtonItem * emailItem;
@property (nonatomic, retain) IBOutlet UISlider * qualitySlider;

-(IBAction) getPhoto:(id) sender;
-(IBAction) emailItem_Pressed:(id)sender;

@end
//
//  CameraViewController.m
//  CMobile
//
//  Created by Andrew Martin on 5/11/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "CameraViewController.h"
#import "JSON.h"
#import "Document.h"
#import "CMobileAppDelegate.h"
```

```
#import "IndexViewController.h"

@interface CameraViewController(private)
-(void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)data;
-(void)uploadImage;
@end


@implementation CameraViewController
@synthesize theimageView, choosePhoto, takePhoto, browseItem, cameraItem,
uploadItem, emailItem, qualitySlider;

-(IBAction) getPhoto:(id) sender {
      UIImagePickerController * picker = [[UIImagePickerController alloc]
init];
      picker.delegate = self;

      if((UIBarButtonItem *) sender == browseItem) {
            picker.sourceType =
UIImagePickerControllerSourceTypePhotoLibrary;
            [self presentModalViewController:picker animated:YES];
      } else if((UIBarButtonItem *) sender == cameraItem) {
            picker.sourceType = UIImagePickerControllerSourceTypeCamera;
            [self presentModalViewController:picker animated:YES];
      } else {
            [self uploadImageAsPost];
      }

      [picker release];
}

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {
      [picker dismissModalViewControllerAnimated:YES];
      theimageView.image = [info
objectForKey:@"UIImagePickerControllerOriginalImage"];

      uploadItem.enabled = YES;
}

-(IBAction)emailItem_Pressed:(id) sender {
      MFMailComposeViewController *picker = [[MFMailComposeViewController
alloc] init];
      picker.mailComposeDelegate = self;

      [picker setSubject:@"CMobile Email"];

      NSData *imageData = UIImageJPEGRepresentation(theimageView.image,
qualitySlider.value);
      [picker addAttachmentData:imageData mimeType:@"image/jpg"
fileName:@"CMobileImageAttachment.jpg"];

      NSString *emailBody = @"CMobile body message";
```

```
      [picker setMessageBody:emailBody isHTML:YES];

      [self presentModalViewController:picker animated:YES];
      [picker release];
}

-(void)mailComposeController:(MFMailComposeViewController*)controller
didFinishWithResult:(MFMailComposeResult)result error:(NSError*)error {
      if(result == MFMailComposeResultSent) {
            UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Email Sent Sucessfully." message:@""  delegate:self
cancelButtonTitle:nil otherButtonTitles:@"OK", nil];
            [alert show];
            [alert release];
      }

      [self dismissModalViewControllerAnimated:YES];
}

-(void) uploadImageAsPost {
      IndexViewController *ivc = [[IndexViewController alloc] init];
      [ivc setImage:theimageView.image];

      [self presentModalViewController:ivc animated:YES];
      [ivc release];
}

-(void) closeCurrentModalView {
      [self dismissModalViewControllerAnimated:YES];
}


- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)data
{
      NSString *jsonString = [[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding];

      NSDictionary *results = [jsonString JSONValue];

      [jsonString release];
}

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
            uploadItem.enabled = NO;
    }
    return self;
}

- (void)viewDidLoad {
      if(theimageView.image != nil) {
```

```
            uploadItem.enabled = YES;
      } else {
            uploadItem.enabled = NO;
      }

      NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];

    [super viewDidLoad];
}

-(void)viewDidAppear:(BOOL)animated {
      NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
      self.qualitySlider.hidden = YES;

      self.qualitySlider.value = [defaults
integerForKey:@"uploadQualitySlider"];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {
    [super viewDidUnload];
}


- (void)dealloc {
    [super dealloc];
}


@end

//
//  CLController.h
//  CMobile
//
//  Created by Andrew Martin on 6/21/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>


@interface CLController :  NSObject <CLLocationManagerDelegate> {
      CLLocationManager *locationManager;
}

@property (nonatomic, retain) CLLocationManager *locationManager;

- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
```

```
           fromLocation:(CLLocation *)oldLocation;

- (void)locationManager:(CLLocationManager *)manager
        didUpdateHeading:(CLHeading *)newHeading;

- (void)locationManager:(CLLocationManager *)manager
       didFailWithError:(NSError *)error;

@end

//
//  CLController.m
//  CMobile
//
//  Created by Andrew Martin on 6/21/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "CLController.h"

@implementation CLController

@synthesize locationManager;

- (id) init {
    self = [super init];
    if (self != nil) {
        self.locationManager = [[[CLLocationManager alloc] init]
autorelease];
        self.locationManager.delegate = self; // send loc updates to
myself
    }
    return self;
}

- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation
{
}

- (void)locationManager:(CLLocationManager *)manager
        didUpdateHeading:(CLHeading *)newHeading
{
}

- (void)locationManager:(CLLocationManager *)manager
didFailWithError:(NSError *)error
{
}

- (void)dealloc {
    [self.locationManager release];
    [super dealloc];
```

```
}


@end

//
//  CMobileAppDelegate.h
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Layouts.h"
#import "Document.h"
#import "DataManager.h"
#import "CLController.h"

@interface CMobileAppDelegate : NSObject <UIApplicationDelegate> {

    UIWindow *window;
    UINavigationController *navigationController;
      UIViewController *baseViewController;
      UITabBarController *homeController;
      UINavigationController *settingsNavigationController;
      UIViewController *rootWindowViewController;
      UIViewController *indexViewController;
      IBOutlet UIBarButtonItem *doneButton;
      IBOutlet NSMutableArray *subViewControllers;
      NSDictionary *globalLayouts;
      Document *documentToIndex;
      DataManager *dataManager;
      CLController *locationController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UINavigationController
*navigationController;
@property (nonatomic, retain) IBOutlet UIViewController
*baseViewController;
@property (nonatomic, retain) IBOutlet UITabBarController *homeController;
@property (nonatomic, retain) IBOutlet UINavigationController
*settingsNavigationController;
@property (nonatomic, retain) IBOutlet UIViewController
*rootWindowViewController;
@property (nonatomic, retain) IBOutlet UIViewController
*indexViewController;
@property (nonatomic, retain) IBOutlet NSMutableArray *subViewControllers;
@property (nonatomic, retain) IBOutlet UIBarButtonItem *doneButton;
@property (nonatomic, retain) NSDictionary *globalLayouts;
@property (nonatomic, retain) Document *documentToIndex;
@property (nonatomic, retain) DataManager *dataManager;
@property (nonatomic, retain) CLController *locationController;
```

```
- (IBAction)done_Clicked:(id)sender;

-(NSDictionary*)getLayouts;
-(void)setLayouts:(NSDictionary *)dictionary;

-(Document*) getDocumentToIndex;
-(void) setDocumentToIndex:(Document*) document;

-(NSMutableArray *) getSubViewControllers;
-(void) setCurrentsubViewController: (UIViewController*) viewController;
-(UIViewController*) getCurrentSubViewController;

-(CLController*) getLocationController;

//other methods
-(void)moveToMain;

@end


//
//  CMobileAppDelegate.m
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "CMobileAppDelegate.h"
#import "LoginViewController.h"
#import "IASKAppSettingsViewController.h"
#import "IndexViewController.h"

@implementation CMobileAppDelegate

@synthesize window;
@synthesize navigationController;
@synthesize baseViewController;
@synthesize homeController;
@synthesize settingsNavigationController;
@synthesize doneButton;
@synthesize subViewControllers;
@synthesize rootWindowViewController;
@synthesize globalLayouts;
@synthesize documentToIndex;
@synthesize dataManager;
@synthesize locationController;


#pragma mark -
#pragma mark Application lifecycle
```

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

      dataManager = [[DataManager alloc] init];

      locationController = [[CLController alloc] init];
      [locationController.locationManager startUpdatingLocation];
      [locationController.locationManager startUpdatingHeading];

      NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
      NSLog(@"wsdlTextfield:%@", [defaults
stringForKey:@"wsdlTextField"]);
      if([[defaults stringForKey:@"wsdlTextField"]
isEqual:@"http://construction-imaging.com/ContentManager/CMSerivce.svc"])
{
            [self
setCurrentSubViewController:settingsNavigationController];
      } else {
            [self setCurrentSubViewController:baseViewController];
      }

    [self.window makeKeyAndVisible];

    return YES;
}

-(void)loadViewController:(UIViewController*) viewController {
      [[self subViews] addObject:viewController];
      [self.window addSubview:viewController.view];
}

-(NSDictionary*)getLayouts {
      return globalLayouts;
}

-(void)setLayouts:(NSDictionary*)dictionary {
      if(dictionary != globalLayouts) {
            [dictionary retain];
            [globalLayouts release];
            globalLayouts = dictionary;
      }
}

-(Document*) getDocumentToIndex{
      return documentToIndex;
}

-(void) setDocumentToIndex:(Document*) document{
      [document retain];
      [documentToIndex release];
      documentToIndex = document;
}

-(NSMutableArray*) getSubViewControllers {
```

```
	if(subViewControllers == nil) {
		subViewControllers = [[NSMutableArray alloc] init];
		[subViewControllers retain];
	}

	return subViewControllers;
}

-(void) setCurrentSubViewController: (UIViewController*) viewController {
	NSMutableArray *vcs = [self getSubViewControllers];

	[viewController retain];

	if([vcs count] != 0) {
		[subViewControllers removeLastObject];
	}

	[subViewControllers addObject:viewController];

	[self.window addSubview:viewController.view];
}

-(UIViewController*) getCurrentSubViewController{
	[subViewControllers objectAtIndex:0];
}

-(CLController*) getLocationController {
	return locationController;
}

- (void)applicationWillResignActive:(UIApplication *)application {
    /*
     Sent when the application is about to move from active to inactive
state. This can occur for certain types of temporary interruptions (such
as an incoming phone call or SMS message) or when the user quits the
application and it begins the transition to the background state.
     Use this method to pause ongoing tasks, disable timers, and throttle
down OpenGL ES frame rates. Games should use this method to pause the
game.
     */
}


- (void)applicationDidEnterBackground:(UIApplication *)application {
    /*
     Use this method to release shared resources, save user data,
invalidate timers, and store enough application state information to
restore your application to its current state in case it is terminated
later.
     If your application supports background execution, called instead of
applicationWillTerminate: when the user quits.
     */
}
```

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    /*
     Called as part of  transition from the background to the inactive
state: here you can undo many of the changes made on entering the
background.
     */
}


- (void)applicationDidBecomeActive:(UIApplication *)application {
    /*
     Restart any tasks that were paused (or not yet started) while the
application was inactive. If the application was previously in the
background, optionally refresh the user interface.
     */
}


- (void)applicationWillTerminate:(UIApplication *)application {
    /*
     Called when the application is about to terminate.
     See also applicationDidEnterBackground:.
     */
}


#pragma mark -
#pragma mark Memory management

- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {
    /*
     Free up as much memory as possible by purging cached data objects
that can be recreated (or reloaded from disk) later.
     */
}

- (IBAction)done_Clicked:(id)sender {
      [self loadViewController:baseViewController];
}

-(void) moveToMain {
      [self setCurrentSubViewController:homeController];
}

-(void) removeIndexViewFromHomeController {
      [indexViewController.view removeFromSuperview];
}

-(void)logout {
      [baseViewController setPerformAutoLogin:false];

      [self setCurrentSubViewController:baseViewController];
```

```
      [baseViewController isLoading:true];
      [baseViewController GetDataSources];
}


- (void)dealloc {
      [navigationController release];
      [window release];
      [dataManager release];
      [super dealloc];
}


@end


//
//  DataManager.h
//  CMobile
//
//  Created by Andrew Martin on 6/21/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Document.h"

@interface DataManager : NSObject{
}

@property (nonatomic, retain) id delegate;

-(void) uploadImage: (Document*) document;
-(void) uploadImageAsPost;

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError
*)error;
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)data;
- (void)connectionDidFinishLoading:(NSURLConnection *)connection;

@end

//
//  IndexViewController.h
//  CMobile
//
//  Created by Andrew Martin on 6/2/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "CLController.h"
#import "Document.h"
```

```
#import "HeaderFields.h"
#import "PickerViewController.h"
#import "DatePickerDoneViewController.h"


@interface IndexViewController : UIViewController <UITextFieldDelegate>{
      NSMutableArray *textFields;
      NSMutableArray *dateFields;
      NSMutableArray *labels;
      IBOutlet UINavigationBar *navigationBar;
      IBOutlet UIBarButtonItem *doneButton;
      IBOutlet UIBarButtonItem *uploadButton;
      UITextField *savedDateField;
      HeaderFields *headerFields;
      PickerViewController *pickerViewController;
      DatePickerDoneViewController *datePickerViewController;
      IBOutlet UIActivityIndicatorView *activityIndicator;
      IBOutlet UIProgressView *progressBar;
      UIImage *imageData;
}

@property (nonatomic, retain) NSMutableArray *textFields;
@property (nonatomic, retain) NSMutableArray *dateFields;
@property (nonatomic, retain) NSMutableArray *labels;
@property (nonatomic, retain) IBOutlet UINavigationBar *navigationBar;
@property (nonatomic, retain) IBOutlet UIBarButtonItem *doneButton;
@property (nonatomic, retain) IBOutlet UIBarButtonItem *uploadButton;
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, retain) UITextField *savedDateField;
@property (nonatomic, retain) HeaderFields *headerFields;
@property (nonatomic, retain) PickerViewController *pickerViewController;
@property (nonatomic, retain) DatePickerDoneViewController
*datePickerViewController;
@property (nonatomic, retain) IBOutlet UIActivityIndicatorView
*activityIndicator;
@property (nonatomic, retain) IBOutlet UIProgressView *progressBar;

-(IBAction) doneButton_Clicked:(id)sender;
-(IBAction) uploadButton_Clicked:(id)sender;

-(void) setImage;
@end

//
//  IndexViewController.m
//  CMobile
//
//  Created by Andrew Martin on 6/2/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "IndexViewController.h"
#import "Layout.h"
#import "Layouts.h"
```

```
#import "CMobileAppDelegate.h"
#import "HeaderFields.h"
#import "HeaderField.h"
#import "JSON.h"
#import "Document.h"
#import <CoreLocation/CLLocationManager.h>
#include "TargetConditionals.h"

@implementation IndexViewController

@synthesize textFields, dateFields, navigationBar, image, savedDateField,
headerFields, pickerViewController, activityIndicator, progressBar;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
            [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(receiveNotification:)

name:@"uploadImageSuccessNotification"

object:nil];

            [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(receiveNotification:)

name:@"uploadImageErrorNotification"

object:nil];

            [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(receiveNotification:)

name:@"uploadImageDataSentNotification"

object:nil];

            [self addTextFields];
      }
    return self;
}

-(void)addTextFields {

}

-(BOOL)datePickerDone:(id)sender {
}

-(BOOL)pickerViewDone:(id)sender {
```

```
}

-(void)textFieldShouldReturn:(UITextField*) sender {
      [sender resignFirstResponder];
}

- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField {
}

- (void)viewDidLoad {
      progressBar.progressViewStyle = UIProgressViewStyleBar;

    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {
    [super viewDidUnload];
}

-(void) viewDIdAppear:(BOOL)animated {
      uploadButton.enabled = YES;
      doneButton.enabled = YES;
}

-(Document*) getNewDocument:(NSData*) imageData {

}


-(Document*) getDocument:(NSData*) imageData {
}

-(IBAction) doneButton_Clicked:(id)sender {
      [self.parentViewController dismissModalViewControllerAnimated:true];
}

-(IBAction) uploadButton_Clicked:(id)sender {
}

- (void)pickerChanged:(id)sender
{
      savedDateField.text = [[[[sender date] description]
componentsSeparatedByString: @" "] objectAtIndex:0];
}

- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation
{
```

```
}

- (void)locationManager:(CLLocationManager *)manager
        didFailWithError:(NSError *)error
{

}

- (void) receiveNotification:(NSNotification *) notification
{

}

-(void) setImage:(UIImage*) i {
      [i retain];
      [image release];

      image = i;
}


- (void)dealloc {
      [textFields release];
      [dateFields release];
      [headerFields release];
      [labels release];
      [pickerViewController release];
      [datePickerViewController release];
      [image release];

      [[NSNotificationCenter defaultCenter] removeObserver:self];
}


@end

//
//  LoginViewController.h
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Layouts.h"

typedef enum {
}AlertToShow;

@interface LoginViewController : UIViewController {
      IBOutlet UITextField *dataSourceTextField;
      IBOutlet UITextField *usernameTextField;
```

```
	IBOutlet UITextField *passwordTextField;
	IBOutlet UIButton *loginButton;
	NSMutableData *responseData;
	NSURLConnection *GetDataSourcesConnection;
	NSURLConnection *LoginConnection;
	IBOutlet UIActivityIndicatorView *activityIndicator;
	Layouts *globalLayoutsCopy;
	BOOL performAutoLogin;
	AlertToShow alertMessage;

}

- (IBAction) loginButton_Clicked:(id)sender;

@property (retain,nonatomic) IBOutlet UITextField *dataSourceTextField;
@property (retain,nonatomic) IBOutlet UITextField *usernameTextField;
@property (retain,nonatomic) IBOutlet UITextField *passwordTextField;
@property (retain,nonatomic) IBOutlet UIButton *loginButton;
@property (retain,nonatomic) IBOutlet UIImageView *backgroundImageView;
@property (retain,nonatomic) IBOutlet UIActivityIndicatorView
*activityIndicator;
@property (retain, nonatomic) Layouts *globalLayoutsCopy;
@property AlertToShow alertMessage;
@property BOOL performAutoLogin;

@end

//
//  LoginViewController.m
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "LoginViewController.h"
#import "HomeViewController.h"
#import "JSON.h"
#import "Layouts.h"
#import "CMobileAppDelegate.h"

@interface LoginViewController(private)
-(void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)data;
-(void)connection:(NSURLConnection *)connection didFailWithError:(NSError
*)error;
-(void)GetLayouts;
@end

NSUserDefaults *defaults;

@implementation LoginViewController
```

```
@synthesize dataSourceTextField, usernameTextField, passwordTextField,
loginButton, backgroundImageView, activityIndicator, performAutoLogin,
alertMessage;
@synthesize globalLayoutsCopy;

- (id) init {
      self = [super init];
      if(self != nil) {
            self.globalLayoutsCopy =  [[UIApplication
sharedApplication].delegate getLayouts];
            self.title = NSLocalizedString(@"Construction Imaging Mobile
Login", @"");
            [self.view addSubview:activityIndicator];

            NSLog(@"init");
      }

      return self;
}

- (IBAction) loginButton_Clicked:(id) sender {
      [self login];
}

-(void) login {
      [self isLoading:true];

      NSString *urlString;

      NSURL *url = [NSURL URLWithString:urlString];
      NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url];

      LoginConnection = [[NSURLConnection alloc] initWithRequest:request
delegate:self];

      [LoginConnection release];
      [request release];
}

- (void)didReceiveMemoryWarning {
      [super didReceiveMemoryWarning];
}

- (void) dealloc {
      [dataSourceTextField release];
      [usernameTextField release];
      [passwordTextField release];
      [loginButton release];
      [backgroundImageView release];
      [super dealloc];
}

-(void) awakeFromNib {
      defaults = [NSUserDefaults standardUserDefaults];
```

```
}

-(void)viewDidAppear:(BOOL)animated {
      [self populateFields];
      performAutoLogin = true;
      [self GetDataSources];
}

-(void) populateFields {
      if([defaults boolForKey:@"toggleSwitchRemember"]){
            usernameTextField.text = [defaults
stringForKey:@"usernameText"];
            passwordTextField.text = [defaults
stringForKey:@"passwordText"];
      }
}

// Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];

      [self isLoading:true];
}

-(void) GetDataSources {
      NSString *urlString;

      NSURL *url = [NSURL URLWithString:urlString];
      NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:5.0];

      GetDataSourcesConnection = [[NSURLConnection alloc]
initWithRequest:request delegate:self];

      [GetDataSourcesConnection release];
      [request release];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)data
{
      [self isLoading:false];

      NSString *jsonString = [[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding];
      NSDictionary *results = [jsonString JSONValue];

      if(connection == GetDataSourcesConnection){
            if(results == NULL) {
                  UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Could not establish a connection." message:[NSString
stringWithFormat:@"%@ cannot be found. Please edit CMobile's URL setting
```

```
endpoint and try again.", [defaults stringForKey:@"wsdlTextField"]]
delegate:self cancelButtonTitle:@"Retry" otherButtonTitles:nil];
                  alertMessage = CouldNotEstablishConnection;
                  [alert show];
                  [alert release];
            }
            else if([jsonString rangeOfString:@"<title>The resource cannot
be found.</title>"].location != NSNotFound) {
                  UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"The service endpoint cannot be found." message:[NSString
stringWithFormat:@"%@ cannot be found. Please edit CMobile's URL setting
endpoint and try again.", [defaults stringForKey:@"wsdlTextField"]]
delegate:self cancelButtonTitle:@"Retry" otherButtonTitles:nil];
                  alertMessage = ServiceEndpointNotFound;
                  [alert show];
                  [alert release];

                  [self isLoading:false];
            } else {
                  dataSourceTextField.text = [[results objectForKey:@"d"]
objectAtIndex:0];
                  [self isLoading:false];

                  if(performAutoLogin && [defaults
boolForKey:@"toggleSwitchAutoLogin"] && [[usernameTextField text] length]
> 0 && [[passwordTextField text] length] > 0){
                        [self login];
                  }
            }
      } else if(connection == LoginConnection){
            NSString *returnValue = [(NSString *) results
objectForKey:@"d"];

            if(returnValue) {
                  [self GetLayouts];
            } else {
                  NSString *exceptionMessage = [results
objectForKey:@"Message"];

                  if([jsonString rangeOfString:@"HTTP Error 503."].location
!= NSNotFound) {
                        UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Login Failed." message:@"Unable to contact the service."
delegate:self cancelButtonTitle:@"Close" otherButtonTitles:nil];
                        alertMessage = LoginServiceFailure;
                        [alert show];
                        [alert release];
                  }
                  else if([exceptionMessage rangeOfString:@"Invalid user
name or password."].location != NSNotFound) {
                        UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Login Failed." message:@"Invalid Username or Password."
delegate:self cancelButtonTitle:@"Close" otherButtonTitles:nil];
                        alertMessage = InvalidLogin;
```

```
                        [alert show];
                        [alert release];
                    }
                }
            }

            [jsonString release];
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError
*)error{
      if(connection == GetDataSourcesConnection) {
            if([[error localizedDescription] isEqualToString:@"The request
timed out."]) {
                  UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Cannot load data sources." message:@"The request timed
out." delegate:self cancelButtonTitle:nil otherButtonTitles:@"Retry",
nil];
                  alertMessage = CouldNotEstablishConnection;
                  [alert show];
                  [alert release];
            } else if([[error localizedDescription] isEqualToString:@"The
Internet connection appears to be offline."]) {
                  UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Cannot load data sources." message:@"The Internet
connection appears to be offline." delegate:self cancelButtonTitle:nil
otherButtonTitles:@"Retry", nil];
                  alertMessage = CouldNotEstablishConnection;
                  [alert show];
                  [alert release];
            }

            [self isLoading:false];
      }
}

-(void) alertView:(UIAlertView*)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex {
      if(alertMessage == CouldNotEstablishConnection || alertMessage ==
ServiceEndpointNotFound) {
            [self isLoading:true];
            [self GetDataSources];
      } else {

      }
}

- (void) isLoading: (bool) b {
      if(b) {
            [activityIndicator startAnimating];
            dataSourceTextField.enabled = NO;
            usernameTextField.enabled = NO;
            passwordTextField.enabled = NO;
            loginButton.enabled = NO;
```

```
	} else {
		[activityIndicator stopAnimating];
		dataSourceTextField.enabled = YES;
		usernameTextField.enabled = YES;
		passwordTextField.enabled = YES;
		loginButton.enabled = YES;
	}
}

-(void)GetLayouts{
	NSString *urlString;
	NSURL *url = [NSURL URLWithString:urlString];
	NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url];
	NSURLResponse *response;
	NSError *error = nil;

	NSData *data = [NSURLConnection sendSynchronousRequest:request
returningResponse:&response error:&error];

	NSString *jsonString = [[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding];

	Layouts *layouts = nil;
	layouts = [jsonString JSONValue];


	CMobileAppDelegate *mainDelegate;
	mainDelegate = (CMobileAppDelegate *) [[UIApplication
sharedApplication]delegate];

	[mainDelegate setLayouts:layouts];

	[mainDelegate moveToMain];

	[request release];
	[jsonString release];
}

-(BOOL)textFieldShouldReturn:(UITextField*)textField
{
	[textField resignFirstResponder];

	return YES;
}

- (void)viewDidUnload {
    [super viewDidUnload];

	[dataSourceTextField release];
	dataSourceTextField = nil;
	[usernameTextField release];
	usernameTextField = nil;
	[passwordTextField release];
	passwordTextField = nil;
```

```
      [loginButton release];
      loginButton = nil;
      [backgroundImageView release];
      backgroundImageView = nil;
}@end

//
//  DatePickerViewController.h
//  CMobile
//
//  Created by Andrew Martin on 6/24/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>


@interface PickerViewController : UIViewController <UIPickerViewDelegate,
UIPickerViewDataSource> {
      IBOutlet UIBarButtonItem *cancelButton;
      IBOutlet UIBarButtonItem *doneButton;
      IBOutlet UIPickerView *myPicker;
      IBOutlet UIDatePicker *datePicker;
      NSMutableArray *myPickerArray;
      NSString *selectedValue;

}

@property (nonatomic, retain) IBOutlet UIBarButtonItem *cancelButton;
@property (nonatomic, retain) IBOutlet UIBarButtonItem *doneButton;
@property (nonatomic, retain) IBOutlet UIPickerView *myPicker;
@property (nonatomic, retain) IBOutlet UIDatePicker *datePicker;
@property (nonatomic, retain) NSMutableArray *myPickerArray;
@property (nonatomic, retain) NSString *selectedValue;

-(IBAction) doneButton_Pushed:(id)sender;
-(IBAction) cancelButton_Pushed:(id)sender;

-(void)addItem:(NSString*)item;
-(void)isDatePicker:(BOOL) isDate;

-(NSString*) getSelectedValue;

@end

//
//  DatePickerViewController.m
//  CMobile
//
//  Created by Andrew Martin on 6/24/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "PickerViewController.h"
```

```
@implementation PickerViewController
@synthesize myPicker, myPickerArray, datePicker;

-(IBAction) doneButton_Pushed:(id)sender
{
}

-(IBAction) cancelButton_Pushed:(id)sender
{
}

-(NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
      return 1;
}

-(NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger) components  {
      return [myPickerArray count];
}

-(NSString*) pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)
row forComponent:(NSInteger) component {
      return [myPickerArray objectAtIndex:row];
}

-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger) row
inComponent:(NSInteger) component {
      selectedValue = [[myPickerArray objectAtIndex:row] description];
}

-(void)addItem:(NSString*)item {
      if(!myPickerArray) {
            myPickerArray = [[NSMutableArray alloc] init];
      }

      [myPickerArray addObject:item];

      if([selectedValue isEqual:@""]) {
            selectedValue = [[NSString alloc] initWithString:[item
description]];
      }
}

-(void) isDatePicker:(bool) isDate {
      if(isDate) {
            [myPicker removeFromSuperview];
      } else {
            [datePicker removeFromSuperview];
      }
}

-(NSString*) getSelectedValue {
```

```
      return selectedValue;
}
- (void)viewDidLoad {
      selectedValue = [[NSString alloc] initWithString:@""];

    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (void)viewDidUnload {
      selectedValue = [[NSString alloc] initWithString:@""];

    [super viewDidUnload];
}


- (void)dealloc {
      [selectedValue release];
      [myPickerArray release];

    [super dealloc];
}


@end

//
//  SettingsViewController.h
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "IASKAppSettingsViewController.h"

@interface SettingsViewController : UIViewController
<IASKSettingsDelegate, UITextViewDelegate> {
      IASKAppSettingsViewController *appSettingsViewController;
}

@property (nonatomic, retain) IASKAppSettingsViewController
*appSettingsViewController;

@end
//
//  SettingsViewController.m
//  CMobile
//
//  Created by Andrew Martin on 3/16/11.
```

```
//  Copyright 2011 Construction Imaging. All rights reserved.
//

#import "SettingsViewController.h"

#import "IASKSpecifier.h"
#import "IASKSettingsReader.h"

#import "CustomViewCell.h"


@implementation SettingsViewController

@synthesize appSettingsViewController;

- (IASKAppSettingsViewController*)appSettingsViewController {
      if (!appSettingsViewController) {
            appSettingsViewController = [[IASKAppSettingsViewController
alloc] initWithNibName:@"IASKAppSettingsView" bundle:nil];
            appSettingsViewController.delegate = self;
            [appSettingsViewController setShowCreditsFooter:NO];
      }
      return appSettingsViewController;
}

-(void) viewDidAppear:(BOOL)animated {
      [self setView:self.appSettingsViewController.view];
}

-(void) viewDidDisappear:(BOOL)animated {
      [[NSUserDefaults standardUserDefaults] synchronize];
}

-(void) viewDidUnload {
      [[NSUserDefaults standardUserDefaults] synchronize];
}

#pragma mark -
#pragma mark IASKAppSettingsViewControllerDelegate protocol
-
(void)settingsViewControllerDidEnd:(IASKAppSettingsViewController*)sender
{
    [self dismissModalViewControllerAnimated:YES];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];

      self.appSettingsViewController = nil;
}

- (void)dealloc {
      [appSettingsViewController release];
      appSettingsViewController = nil;
```

```
    [super dealloc];
}
@end
```