lilitab

inspired interactions

LILITAB SDK OVERVIEW

v2.0

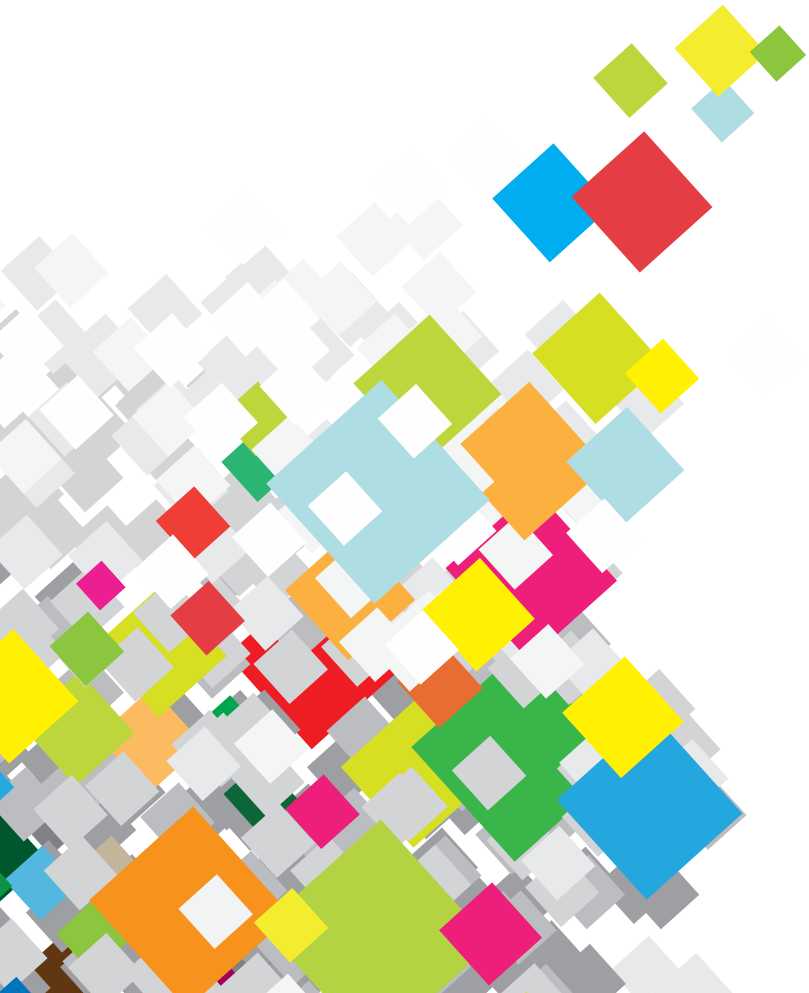

TABLE OF CONTENTS

1. LILITAB SDK OVERVIEW (v2.0) INTRODUCTION

This document provides developers an introduction to the lilitab swipe software development kit (SDK). As an SDK, the contents consist of two starter app Xcode projects enabling iOS developers to begin using the lilitab swipe w/integrated MagneSafe reader from MagTek. In addition, we provide a static library (libLilitabSDKUniversal.a) and header files. The library and header files, along with the EAAccessoryFramework must be included in your project to properly use the lilitab swipe device.

This is not a complete or production-level solution for any user application. The included sample projects provide a starting point to begin using the SDK. The lilitab swipe is a powerful, highly flexible device that includes an intelligent reader containing its own embedded microprocessor.

You must have a working knowledge of iOS, the Apple Xcode development environment (IDE), and Cocoa Touch in order to effectively begin developing native iOS apps. If you plan to integrate with a web page, your team should obviously understand HTML and JavaScript development. You should review the basics of the EAAccessoryFramework for any type of app and understand how to use the `stringByEvaluatingJavaScriptFromString` method if developing a web-based solution.

Finally, you should read through the MagTek SPI MagneSafe V5 Communications Reference Manual making sure you have the most recent version. The manual provides the details on data and command formats used by the reader.

2. GETTING STARTED

These examples presume using an **unencrypted reader head**. Unless specifically requested, all lilitab swipe units are unencrypted.

2.1 Open and build the projects..

To begin, (1) unzip SDK Distribution zip file to your Xcode development Mac computer, (2) change the project settings to reflect your own iOS provisioning profile, (3) build and run the app on your iPad.

The unzipped file contains two Xcode projects, the static library libLilitabUniversal.a, the Lilitab SDK Headers folder and the MagTek communications document. The projects should be labeled LilitabNativeDemo and LilitabWebDemo. Each project already contains the static library and header files.

Open either project folder and double click on the appropriate .xcodeproj file. Next, change the provisioning profile for your device, then build and run the project on your device, immediately pressing the home button to return to the springboard.

Figure 1 shows the results of building both sample projects and installing on your iPad. Your location and springboard layout may be different.

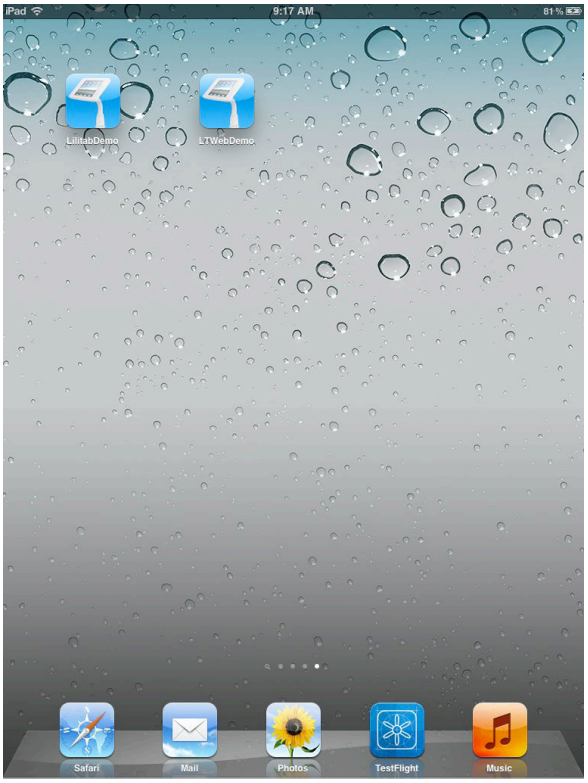

Figure 1. App icons on the iPad springboard

2.2 Test the Native iOS Project

Before inserting the iPad into the swipe unit cradle, it is a good idea to run each of the sample projects to get an idea of what you will be seeing.

First, tap the icon labeled LilitabDemo to start the native iOS demo application. You will see the screen depicted in figure 2 which displays a template of information that might represent a sample item for sale by a retailer. A UIButton object labeled "Buy Now" activates a modal screen that allows the user to swipe their credit card.

Tap the "Buy Now" button to see the modal screen shown in figure 3.

Figure 2. Native iOS Demo app initial screen

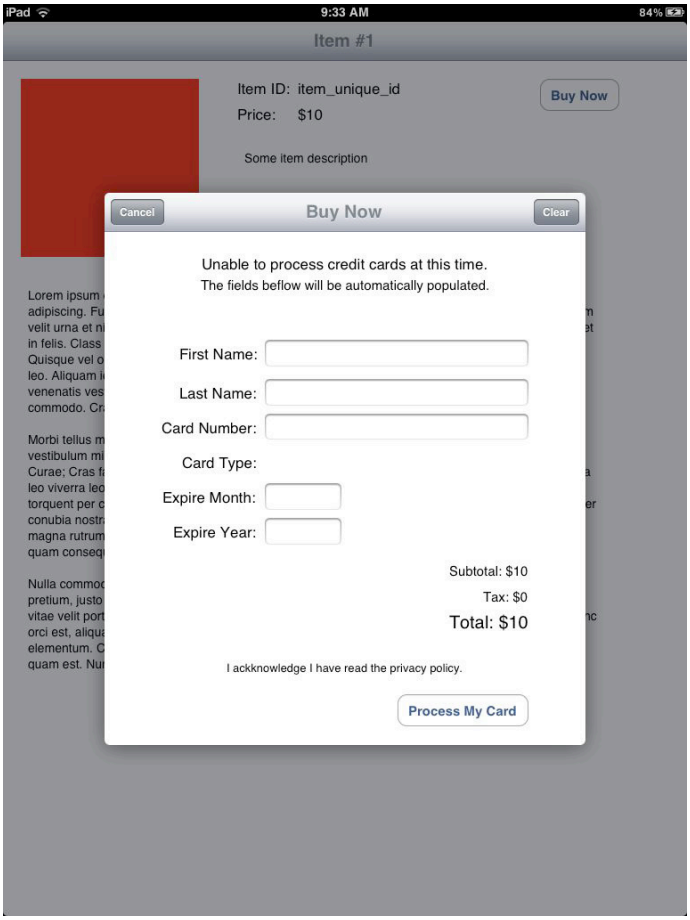

Figure 3. Native iOS Demo app payment screen NOT INSTALLED in lilitab swipe

Note that, because the iPad has not been inserted into the lilitab swipe unit, the message "Unable to process credit cards at this time" is displayed at the top of the modal dialog. If you were to now insert the iPad, with the app still running, into the lilitab swipe, the modal screen will change to that shown in figure 4.

Now, you can see that the message in the modal dialog has changed to indicate that you can swipe your card through the reader. If you do not see this message, then remove and reattach the iPad making sure that you provide a good connection to the 30-pin dock connector. The lilitab swipe unit does not need to be powered to perform this testing. Power is provided by the iPad to the lilitab swipe electronics.

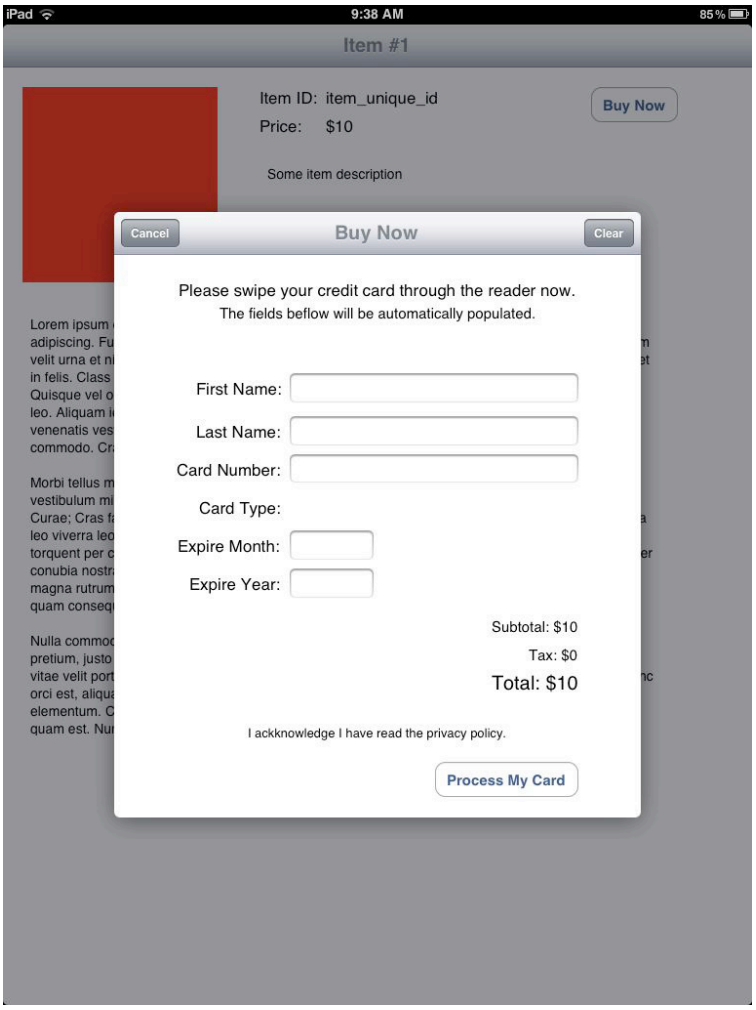

Figure 4. Native iOS Demo app payment screen INSTALLED in lilitab swipe

If you have a credit card available, go ahead and swipe it through the reader making sure that the card stripe is oriented properly—facing away from the iPad. You will see your data displayed similarly to that shown in figure 5.

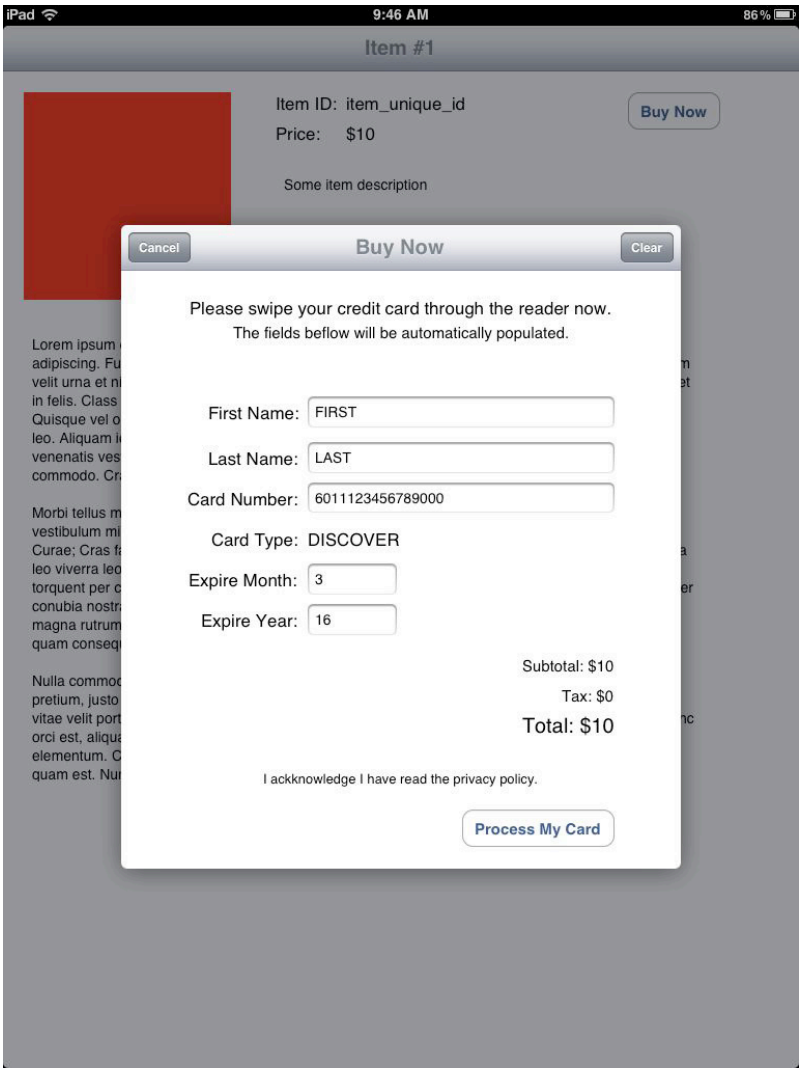

Figure 5. Native iOS Demo Sample Card Swipe

The native iOS Demo Project uses the delegate method

```
- (void)accessoryDidPassRawDataOnly:(NSString *)rawData;
```

to capture the complete stripe data from the card. Within this method, there exists sample lines of code to perform basic parsing of the data to the fields in the modal popover page.

2.3 Test the Web-Based iOS Project

Tap the icon labeled LTWebDemo to start the web-based iOS demo application. You will see the screen in figure 6.

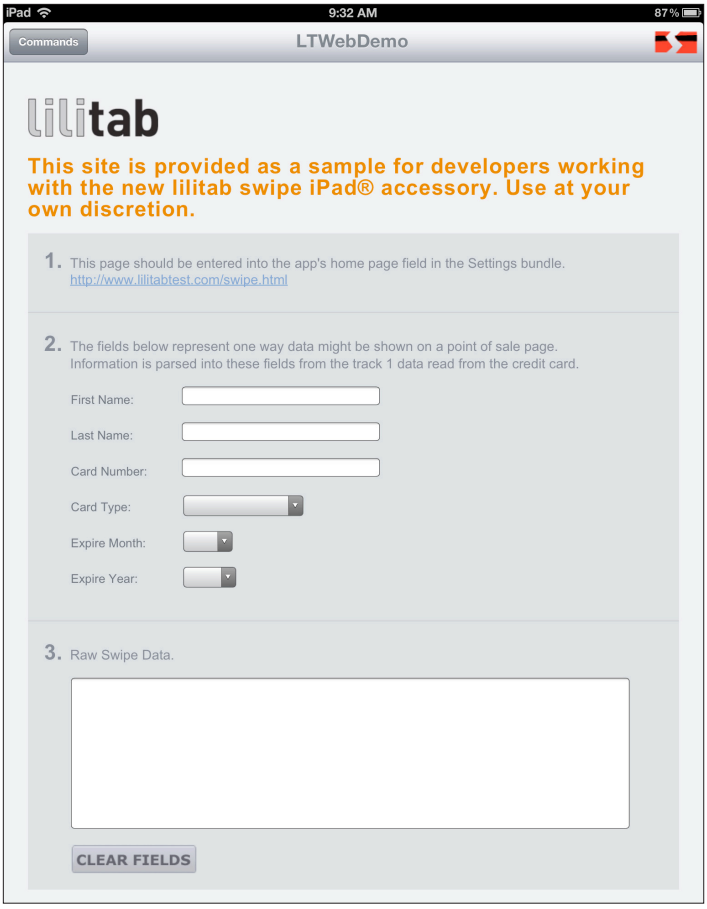

Figure 6. Web-Based Demo app initial screen

While the native iOS demo app was very simple and straightforward, this app is far more versatile and therefore more complicated. LTWebDemo presents a UIWebView in the main view controller along with a toolbar at the top. The bar contains a button on the left that provides access to commands and an icon on the right that indicates the connectivity to the lilitab swipe unit.

If the iPad is not installed in the swipe unit, the icon will depict a red, broken credit card as shown in figure 6. If installed, the icon will be a solid green credit card. Again, when inserting the iPad into the swipe unit, make sure to obtain a solid connection at the dock connector. Be careful not to get paper, lint, or other materials in the area of the connector.

Below the tab bar in figure 6, the UIWebView displays the home page as defined in the Settings. The default page that comes with the demo app is http://www.lilitabtest.com/swipe.html. This is a page that we provide as a starting point that you can use as a reference for your own web page development. This page is designed for testing ISO (Bank/Credit) cards. You can also use the page http://www.lilitabtest.com/swipe-dmv.html for testing AAMVA (Driver's License) cards.

This sample page contains three sections, the first refers to the web page address. The second section is a set of six fields that you might use to parse and display confirmation bank card data to a customer. The third section allows you to display raw data such as complete track strings or parameters returned from commands. Each field is labelled with an HTML ID tag. You should review the web page source code for more details.

If you were to now insert the iPad into the lilitab swipe unit with the app running, and swipe a card, you would see something similar to that shown in figure 7. Note that the solid green card icon appears at the right side of the bar.

Figure 7. Web-Based Demo Sample Swipe

IMPORTANT: The iOS code does not parse or display any data. Parsing and displaying of card data is performed by the JavaScript code in the web page.

Pressing the Clear Fields button removes any data that is currently displayed. Clearing is done by JavaScript in the web page. The IMPORTANT point here is that the main function of the project's Objective-C code is to pass data to a web page.

If you tap on the "Commands" button, you will see a drop-down table view of sample commands as shown in figure 8. In the demo app not all of the sample commands in this view are currently implemented. As the SDK progresses we will implement more of the functionality. However, because of the large number of commands offered by the MagTek reader, developers will need to

implement the command set that they require in their application. Most developers will likely never need to command the reader directly. And of those developers that do need commanding, most will only need to implement a few commands in their app.

We have provided the most basic command set that gives developers a starting point for their own applications. Commanding can be implemented in a variety of ways. The LTWebDemo app uses a command completion block callback architecture.

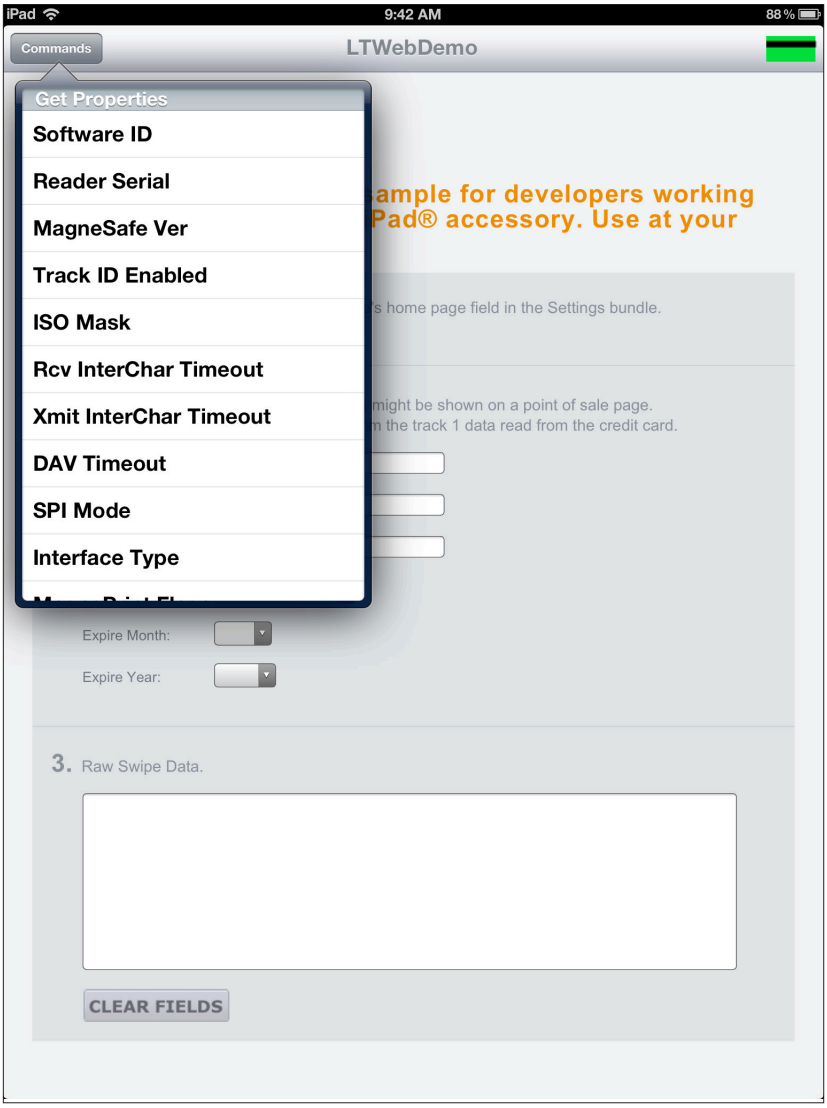

Figure 8. Web-Based Demo Command Display

If you select a command, such as the Software ID of the reader, the information will be appended to the raw data display as shown in figure 9. Only a new swipe or pressing the Clear Fields button clears the raw data display.

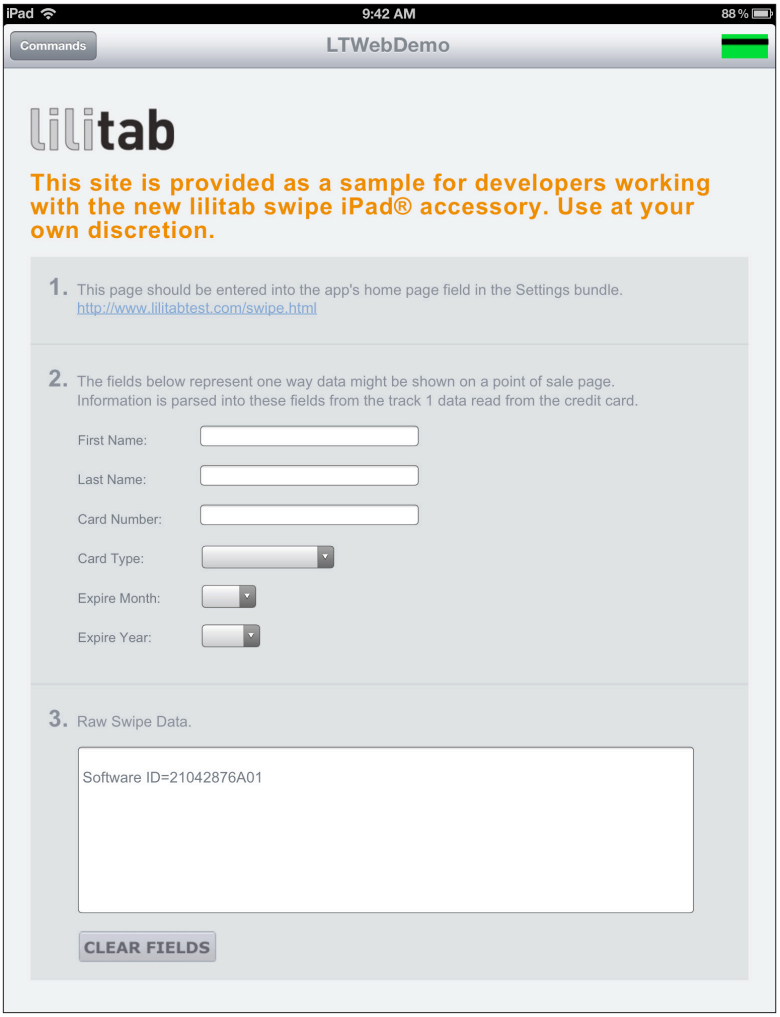

Figure 9. Web-Based Demo Software ID Command Example

Sample Web Demo App Summary

This quick start has shown only the most basic functionality of the web-based demo app as implemented in the delivered SDK.

You absolutely should review the code, primarily the LTViewController.m file, each time you receive an updated SDK delivery. Notes will be provided as to changes, but the bulk of modifications will occur in this file with any exceptions noted.

3. ARCHITECTURE/INTERNALS

The remainder of this document focuses on various aspects of the SDK to aid the developer in using the sample projects as a starting point for their own applications. The following discussion will, except where noted, use the web-based demo project as a reference. Other than code that interfaces with the web page such as the stringByEvaluatingJavaScriptFromString: method, these techniques should work the same in a native iOS application.

This document assumes a basic understanding of the use of Apple's Xcode IDE, structure and organization of apps, Objective-C, the idea of settings and user defaults, delegation, Automated Reference Counting (ARC), blocks, and other skills normal to developing any iOS application.

Two KEY POINTS to take away are:

1. Domain knowledge – you need to have knowledge of the card types you are intending to process. There is no one-size-fits-all solution to handling card data.
2. Process raw data – you must use the accessoryDidPassRawDataOnly: method to retrieve the raw track data and parse it as your application requires.

We have found that the best model for customers to implement is one that passes raw data to a JavaScript method on their web page. The native iOS code can stay consistent over time, requiring fewer updates. This is especially important if you are choosing to distribute your app through the App Store.

The JavaScript method, because it is your web source and under your control, can be changed or updated at any time without review by Apple. So, if a new card type that you need to process is identified, you change the web-based JavaScript, for example, with no changes to the Objective-C native iOS code.

As with any complicated system, there are many ways that the architecture may be represented. Figure 10 depicts the major system elements and their interconnections. Boxes represent "hardware" elements such as the web server, the iPad, or other electronics. Solid lines represent communications between the hardware elements.

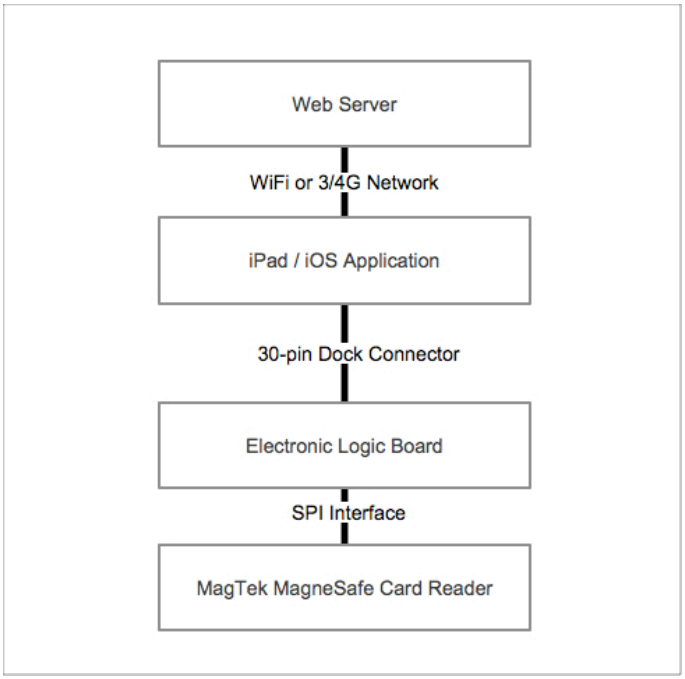

Figure 10. Basic Web-Based App Architecture

At the very top, the web server provides the HTML code for the page layout that displays in the UIWebView window of the application. The server delivers the page to your iPad via a local WiFi connection or your phone connection if you have one. Also at this level is any JavaScript you might have to perform other functionality such as parsing.

Using the Apple Xcode IDE you build and load the application onto your iPad. The app uses a UIWebView object to load and display the web page similar to a web browser such as mobile Safari. Using a hybrid-native/web app allows the use of the software commands needed to get data from external hardware such as the MagTek MagneSafe card reader.

The bottom two boxes in figure 10 (Electronic Logic Board and MagTek MagneSafe Card Reader) represent the external accessory subassembly. The reader is connected to the logic board via a standard Serial Peripheral Interface, or SPI bus.

An *external accessory* is a device to which the iPad can be connected. Electronically it can be connected wirelessly through a mechanism such as Bluetooth, or in the lilipad swipe, via the physical 30-pin dock connector.

External accessories (e.g. card readers) communicate with iOS applications using the EAAccessoryFramework. While we try to isolate you from the details to make your job easier, you may want to become familiar with the EAAccessoryFramework to understand what is going on at the lower levels. You can find documentation on understanding these frameworks at Apple. For the most part, the source code provides a great start on what you need to talk to this (the MagTek reader) accessory.

The iOS app communicates with the logic board that contains a small microcontroller. Commands directed to the MagTek reader are "passed-through" the logic board to the reader. How you format commands for the reader is described in the MagTek document.

When you create a command in your app, you will set the destination as the reader. Like the logic board, the reader also contains an intelligent microcontroller.

4. SOFTWARE/FILE STRUCTURE

Again, the software discussed in this section refers to the web-based iOS application that you will build using the Xcode IDE. It also does not refer to any firmware located on the logic board

Figure 11 shows the LilitabWebDemo Xcode project hierarchy.

There are three top level folders: (1) LilitabWebDemo contains the project source code as well as the SDK library and header files, (2) the Frameworks folder contains the iOS frameworks needed to build the project, and (3) the products folder. Note that the EAAccessoryFramework is included in the Frameworks folder. If you create your own app from scratch, you will need to make that addition. Also note that the libLilitabSDKUniversal.a library is inside the LilitabWebDemo folder. You may want to put this into the Frameworks folder depending on your own organization. Finally, at the top level are the two icon files: normal and retina display. You can change your hierarchy to suit your local guidelines.

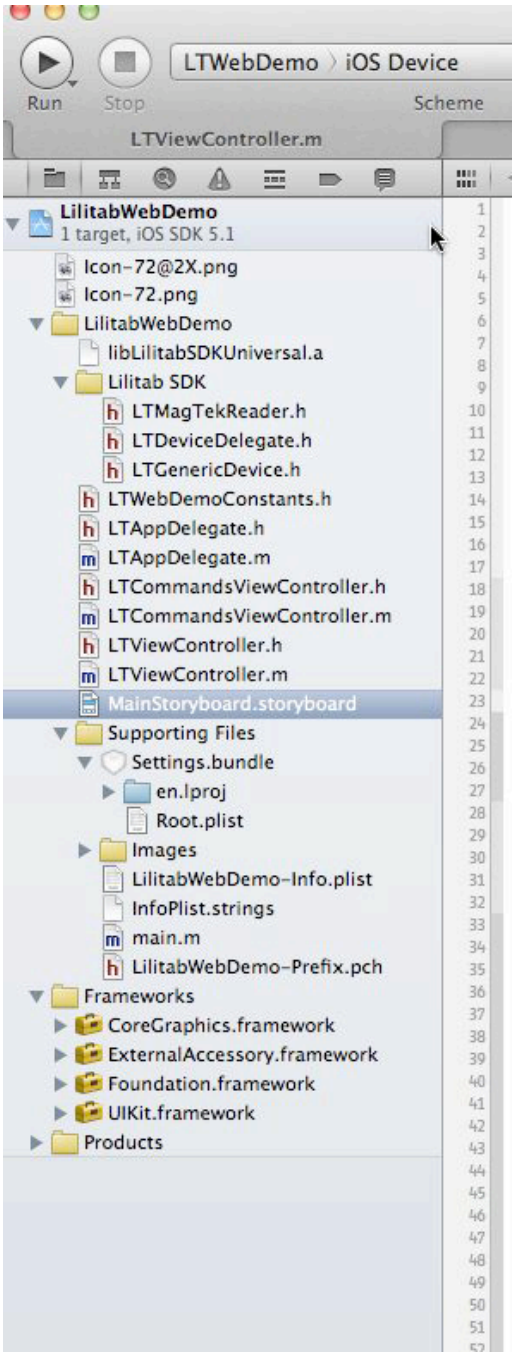

Figure 11. Xcode Project Hierarchy for LilitabWebDemo

Inside the LilitabWebDemo folder is the LilitabSDK folder containing the header files needed by your project. A single header file, LTMagTekReader.h provides access to the reader. You will also see LTGenericDevice.h which is the basis of the reader header. LTGenericDevice provides all the common functionality access and just about everything you'll need is there. However, should a different reader type be provided later on, that reader may have specific functions that extend the generic device.

The main project files are the AppDelegate, the LTViewController, and the LTCommandsViewController. Note that the app was created using storyboards as indicated by the MainStoryboard.storyboard file. If you are not familiar with storyboards, you may want to review them at the Apple developer web site.

As should be immediately apparent, this is a very simple app from an iOS architectural standpoint. The LTViewController contains a toolbar at the top and a UIWebView below extending to the bottom of the iPad view area. Within the toolbar, to the left is a UIButton that causes the commands view controller, a subclass of UITableViewController to be displayed.

Within the Supporting Files folder are two files that you should be aware of. The first is the Settings bundle, in particular the Root.plist file. This is where the user can change settings outside the app from the springboard's Settings app. As delivered, the web-based demo app provides the Settings options that are shown in figure 12.

There are two Settings groups: Web Settings and JavaScript Method Names.

The most important field is the Home Page under Web Settings. Here, you can change the web page you wish to load when the app launches. If you have clients of your own, each of which have a different point of sale (POS) web page, for example, you use this method to allow them to set which page is to be shown when starting the app.

The JavaScript Method Names grouping is where you set the JavaScript function name that you wish to be performed when a certain action occurs. The web demo app provides four default conditions: When a card is swiped, when a parameter is returned from executing a Get Parameter command, when the reader is connected, and when the reader is disconnected. Review the HTML code JavaScript for more information. You may want to add your own specific functions as well.

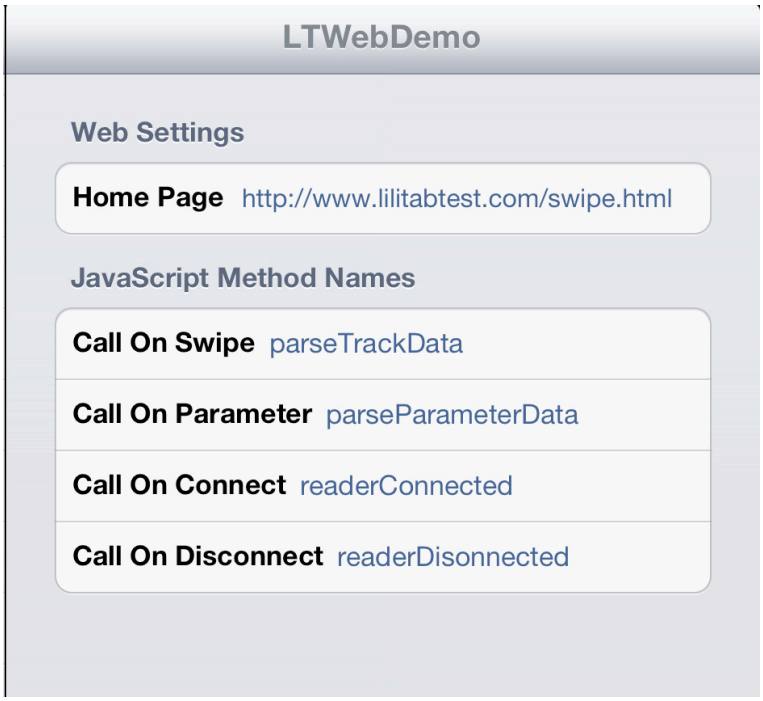

Figure 12. LilitabWebDemo Settings Options

The second file of interest in the Supporting Files folder is the LilitabWebDemo-Info.plist. Having built and deployed apps to devices before, you should understand the importance of this file.

The main thing of concern here is the supported external accessory protocols. To use the lilitab swipe, as with any dock-connected accessory, this must match the protocol string in the device. The current production level swipe units contain the protocol string "com.lilitab.p1." This is the protocol string you should use in your development.

The protocol string defines a connection between the hardware, the app, and the App Store. If your app does not contain the required protocol string, it won't work with the hardware. Also, the protocol string is used to define which apps work with what hardware. If you plugged a clean iPad into a lilitab swipe unit, it will issue an alert and ask if you want to go to the App Store where all apps that use the protocol string found on the hardware will be shown.

Some developers may want to only show their app and not others. This can be done, but requires many more steps in the process. You can consult with your lilitab representative for more information. But, be advised, it will cost more and take more time as it involves a more complicated Apple review process as well as sending verification hardware to Apple that they keep. Basically this means that you have to pay for two additional lilitab swipe units that Apple keeps to test updates to your app.

So now you have a basic understanding of the file structure of the web-based demo project. Before getting into the details of the code, the data flow will be described.

5. DATA FLOW – FROM THE READER TO THE APP

Two types of data flows exist in the system:

- the reader sends **notifications** unilaterally to the iOS application, through the logic board, and
- the reader also sends **responses** as the result of receiving a command from the iOS application.

A credit card swipe initiates a notification to the iOS application that contains credit card track data. The data may be cleartext or encrypted. The application does not send any commands to receive credit card data, but everything is initiated from the reader when a card is swiped. The list below provides a sequential ordering of what goes on when a card is swiped.

1. Customer swipes credit card.
2. MagTek reader decodes the stripe data and raises the Data Available (DAV) line on the logic board.
3. Within the electronics, the DAV line generates an interrupt to the microcontroller and invokes an interrupt service routine (ISR).
4. The logic board ISR sets a flag to indicate that the reader has data ready.
5. In the main program loop of the logic board (NOT the ISR) the number of characters the reader has to send is determined.
6. Still on the logic board, a loop is executed for the number of characters available. One character is loaded from the reader to the logic board, then that character is sent to the iPad. This provides additional security so that there never is more than one character from a customer's card stored on the logic board.
7. In the iOS application, the EAAccessoryFrameworks accesses iOS to load all of the data from the logic board into a buffer. This is done in the libLilitabUniversal binaries and provided to your app via delegate method callbacks.
8. In the iOS application, you implement the accessoryDidPassRawDataOnly: delegate method to process the raw data string, taking care to fully understand not only the card data format, but the ancillary characters that the MagTek reader adds.

If you choose not to operate the MagTek MagneSafe reader in its default mode, you will need to send commands to the reader to change its properties and/or other settings. You will send a command to set the value of a property, and then send a command to read back the property value to verify that it was properly changed.

Even if you intend to use the reader in its default mode, you may wish to get information from the reader such as the reader's software ID or serial number. To get the serial number of the reader, for example, you send a get property command specifying the serial number property to the reader that is then returned to your application.

An important point here is that this is NOT a command and wait operation, i.e. there is no polling for a response. You send a command to the reader to get the parameter of interest. You might implement something like

```
- (void)commandWasSelected:(NSIndexPath *)commandIndexPath
```

and select based on the indexPath that you used to send the command. For example, if you set the GetSoftwareID command in section 0, row 0, you would process command response data for that command by looking for that indexPath. In the web-demo sample project, the indexPath for commands are defined via their placement in the CommandsViewController object in the main storyboard.

6. COMMAND FLOW – FROM THE APP TO THE READER

As described earlier, the web-based demo app provides a button on the left side of the tool bar that you can use to send sample commands to the reader. Three categories of commands are provided: (1) Get Property, (2) Other Get Commands, and (3) Set Commands.

CURRENTLY only the Get Property type of commands are implemented.

Some values such as the reader's security level are not properties but intrinsic to the reader itself.

Tapping the MagTek Commands button displays a popover table view of provided commands. These are NOT all the commands the reader is capable of handling. The default application provides a representative sample of how you would deal with the various types of commands. Just as with how and what data you decide to display from a card swipe, you will need to decide which command you need in your specific application. Refer to the MagTek Communications Reference for more details.

Note: Lilitab LLC can provide assistance such as adding command templates for commands that your organization requires. We can create a semi-custom version of this SDK that has "more of what you need". Contact your Lilitab representative for more details.

The popover section "Get Property" allows you to send a command to retrieve and display the property listed. Several options are provided.

7.0 PROJECT CODE – THE VIEW CONTROLLER

The heart of the project is the LTViewController object. By going through the LTViewController.m file you should be able to understand how to do just about anything needed to read and process your card data.

This section provides a general overview of the LTViewController.m file and points out specifics that you need to be aware of when developing your own application.

This file contains the following methods that you should be aware of:

```
- (void)viewDidLoad –
- (void)viewDidUnload
- (void)gotoHomePage
- (void)prepareForSegue:
- (void)accessoryWasConnected:
- (void)accessoryWasDisconnected
- (void)accessoryDidPassRawDataOnly:
- (void)commandWasSelected:
```

In addition, at the top of the file is the following code snippet:

```
#define USE_UNENCRYPTED_READER 1
```

In the viewDidLoad method this environment variable determines which initialization path is executed. Note that either path initializes the reader in the same manner...the only difference being a defaults variable being set. From either an encrypted or unencrypted reader, you get the complete set of data from the swipe. But, an encrypted reader provides much more data. Review the MagTek Communications Reference for content details. Once you get the data from the reader, you will likely perform different functions depending on your encryption level.

viewDidLoad:

At the bottom of the method, the defaults are synchronized and the homepage convenience method is called to load the home page.

To initialize the reader:

```
_cardReader = [[LTMagTekReader alloc] initWithDelegate:self andProtocolString:@"com.lilitab.
p1"];
```

for either an encrypted or unencrypted reader, as long as it is a MagTek device.

Most of this functionality would be also applicable if you were creating a native iOS app with the exception of loading a web page.

viewDidUnload:
As with any app supporting ARC, set your objects to nil so they can be released. Note also, the statement:

```
[self.cardReader tearDown];
```

This performs all the tear down required by the EAAccessoryFramework and is taken care of by the static library functionality.

goToHomePage:
This is a simple convenience function to load the destination home page, getting the web address from the NSUserDefaults. You will want to make this a secure connection to your web service. You should review all Apple documentation regarding network security to assure your app meets PCI compliance requirements if needed.

Also notice the line:

```
[request setCachePolicy:NSURLRequestReloadIgnoringLocalCacheData];
```

which specifies the cache policy. This line forces the web page to reload each time the app starts. Otherwise, the app may load from a cached page and not see any changes you made on your page.

prepareForSegue:
You should understand the basic purpose of this method if you are familiar with designing using storyboards. Here, we get a weak reference to the popover controller so it can be dismissed once the command has completed. If you are unfamiliar with storyboards, you could implement this method using a different method of your own choosing.

accessoryWasConnected:
This method is called when the iPad and lilitab swipe unit are physically connected. This method is optional, but you may want to use it to be aware of whether a good connection has been made. Here, all we do is set the icon in the tool bar.

accessoryWasDisconnected:
The converse to the previous method, called when the iPad is removed physically from the lilitab swipe, here we again set the icon to be displayed in the tool bar.

accessoryDidPassRawDataOnly:
What data comes from the reader on a card swipe depends on a couple of things. First, in an error condition, you might get something like

```
<MagTek header bytes><T1 SS>E<T1 ES><T2 SS>E<T2 ES><0x0d>
```

which indicates that a error occurred reading both track 1 and track 2. You should recognize that SS and ES indicate the appropriate start sentinel and end sentinel respectively.

For a good read, with an unencrypted reader, you will get the same header and 0x0d trailer, sentinels, and the raw track data partitioned into track 1 and track 2 between the appropriate sentinels. For custom cards you can see any variety of characters, usually with the standard start and end sentinels, but not necessarily. In general, for unencrypted T1 and T2 you should get somewhere around 100 – 130 characters depending on how the card is formatted.

For an encrypted card reader, you will generally get a lot more data, typically near 500 characters.

The exact format of the data returned when using either card reader will depend on how your reader is configured. The MagTek reader is capable of reading 3 tracks of data, but by default passes only track 1 and track 2. To change this you will need to implement the reader commanding features necessary as defined in the MagTek communications document.

The LTViewController.m file's implementation of the accessoryDidPassRawDataOnly: method contains less than a dozen lines of functional code. In essence, all it does is trim off the non-ASCII characters from the start end end of the swipe data, look up

where to pass the data from NSUserDefaults (the JavaScript method to call), and then pass the data to that method.

How you choose to implement your algorithms is up to you. All this sample project does is pass everything to the web page code using the stringByEvaluatingJavaScriptFromString: method. For information on how parsing is actually accomplished, review the HTML and JavaScript code by examining the web page source.

commandWasSelected:
This method is only required if you are using commanding, including reading parameters, in your application.

The organization of the nested if/else structure should be somewhat obvious. The indexPath's section number is used to identify the type of command, here a 0 means a get parameter function. The indexPath's row identifies the command within that section in the command table view controller popover.

Each section of code within this if/else structure calls a specific SDK library method to form and send that command.
For example:

```
//Software ID
  if(commandIndexPath.row == 0) {

     [self.cardReader softwareIDWithCompletionHandler:^(NSString *commandValue) {
     NSString *js = [NSString stringWithFormat:@"%@('\\nSoftware ID=%@');",[defaults
objectForKey:kCALL_ON_PARAMETER_KEY],commandValue];
     [self.webView stringByEvaluatingJavaScriptFromString:js];
            }];
   }
```

implements the call to read the software ID from the MagTek Reader. The method softwareIDWithCompletionHandler is called passing in the completion handler block.

The return value of the get parameter, in this case the software ID, is returned in the NSString pointer commandValue. Within the completion handler is the code to pass the data to UIWebView's current web page.

Finally, at the bottom of this method the popover is dismissed.

8. TYPICAL USE CASES

Because of the flexibility of the lilitab swipe unit, there is no single, one-size-fits-all application or sample that we can provide to make everyone happy. As such, we have endeavored to provide a sample project that includes enough that most use cases can be satisfied.

We believe that most users of our SDK will be processing some type of bank card, e.g. debit card, credit card, student ID with credit, etc. Of those users, some will be developing native iOS apps while others will have an iOS "wrapper" around a UIWebView that passes data to their web service. We feel that this provides the best solution, i.e. the LiltabWebDemo type of solution. Why? Because at last count there were over 155,000 different types of cards in one company's card database. In addition, in one month 20,000 changes (deletions, additions) were made to that database. If you were to, for example, identify the type of card in your code and set something to indicate that this card was a Visa or Discover in your native iOS code, then, to keep up you would have to update your app every month based on the new database changes. Alternatively, you could use a web-based Issuer ID (IID) lookup service from your iOS app, but then you would need to deal with additional PCI security as well as possible network outages.

If you simply pass the complete raw data as read from the card to your web service, then you could make changes to the basic functionality of your processing (parsing, displaying verification data, etc.) without re-issuing your native iOS app. This would be extremely important if you plan to deploy on the App Store. Of course, the choice is yours and there are certainly ways to "make it work" such as using an online database IID lookup service.

Another use case of interest would be processing non-bank cards, such as insurance cards or driver's licenses. In essence, there is really no difference except in how you parse and use the raw data from the card reader. For situations where you need to read track 3 data, you will either have to develop the commanding capability to change the track usage on the reader, or work with your Lilitab representative. We are looking at ways to pre-configure certain functions in the readers we ship with our lilitab swipe units, such as turning on track 3 reading. We hope to have this capability in the near future.

Our suggestion to get started if you feel overwhelmed would be to make a very simple, native iOS app with a single view controller and a single text field for displaying card data. Add the library and header files to your project, remembering also to add the EAAccessoryFramework. Implement the appropriate type of reader initialization, making sure to include the protocol string in your .plist file. Implement the accessoryDidPassRawDataOnly: method and extract the data between the T1 start sentinel and the T2 end sentinel displaying that data to the text field in your view. The most common mistake when trying to do this is that you do not extract just the card data, leaving the non-ASCII prepended MagTek status bytes or the trailing 0x0d CR character. These may not display properly.

9. CONCLUSION

This document provides developers a quick understanding of the sample applications provided in the lilitab-swipe SDK. Neither project is intended as a deliverable application to your customers, but should be used only to provide understanding in how certain things function and as a starting point for your own application. You can certainly cut and paste from the sample project(s) as needed.

You should have a working knowledge of iOS and the Apple Xcode development system. You should review the EAAccessoryFrameworks and know how to use the `stringByEvaluatingJavaScriptFromString` method. You should have domain knowledge of credit card formats and back-end systems. As stated earlier you should understand the structure and organization of apps, Objective-C, the Settings bundle and user defaults, delegation, Automated Reference Counting (ARC), blocks, and other things normal in developing any iOS application.

Finally, you should go through the MagTek SPI MagneSafe V5 Communications Reference Manual making sure you have the most recent version. That manual provides the details on data and command formats used by the reader.

Support

If you need additional technical support, from simple questions all the way to configuring a special version of the SDK or even custom iOS app design support, contact your Lilitab representative.

App Store

To quickly get started, download the **LilySwipe** app from the Apple App Store. NOTE that if you plug your iPad into the lilitab swipe withoug any apps installed, iOS will ask if you want to go to the App Store to get an app that works with the lilitab swipe unit. There may be several apps, written by third parties (other iOS developers) presented for download. Be sure to only select LilySwipe from Lilitab LLC unless you are looking for another specific app to which you have been directed to use.

10. REVISIONS

This section identifies revisions in the SDK, this document, or both. Revisions will be identified as follows
vn.m-p where
n = major SDK software release
m = minor SDK software changes, upgrades and/or bug fixes
(-p) = document only changes - no changes to software

Examples:

v1.1 indicates SDK release 1.1 and a new SDK Overview document
v1.1-1 indicates the same SDK release 1.1 but the SDK Overview document has been modified.
Revisions will be listed starting with the most current release.

REVISIONS:
v2.0 - 9/14/12 - PRODUCTION RELEASE. Formerly accessible parsing methods from the library have been removed. A single reader header, LTMagTekReader.h replaces the LTUnencryptedReader.h and LTEncryptedReader.h. The demo projects have been updated to use the new library.

V1.1 – 7/2/12 – Initial SDK 1.1 and Overview document release