

## XGPS170 SDK Revisions

### Version 2.5

- The code has been updated to support ARC and iOS 7.
- The methods for generating images from weather data are faster, more memory efficient and offer more ways to request data.
- Methods were added to change device settings.
- A bug affecting the rendering of scaled CONUS data was fixed.
- Traffic messages can now be selectively turned on/off.
- Bugs in the decoding of Basic and Long traffic messages were fixed.
- A much more useful demo app for the Full SDK is now included.

### Welcome!

Welcome to developing with the Dual Electronics XGPS170 GPS + ADS-B Receiver, and thank you very much for your interest! This developer's kit contains the information you will need to make an iOS application connect and communicate with the XGPS170.

The goal of this developer's kit is to make app integration as easy as possible. We welcome suggestions on how to improve this SDK in any way.

### About the XGPS170

The XGPS170 is a 978MHz ADS-B In receiver with a WAAS GPS receiver. It is certified to work with all Apple iPod touch®, iPad® and iPhone® devices with Bluetooth®. It will also work with Android®, Windows®, and OS X® devices. However, this SDK only covers integration into iOS® applications.

The XGPS170 can connect to as many as three devices simultaneously: two over Bluetooth and one over USB.

All information received on the ADS-B broadcast is available through the XGPS170 to your app. The device does not restrict access to any ADS-B weather or traffic information.

### Blackjack

The codename for the XGPS170 is **Blackjack** since the device looks a little bit like a deck of cards. So this documentation, the API and the accompanying code will use **Blackjack** interchangeably with **XGPS170**.

Throughout this documentation & the code, the term **iOS devices** will be used to collectively refer to the iPad, iPod touch and iPhone devices.

### PLEASE READ: Apple-related Requirements Affecting You

Before you submit your app to the iTunes store for review, you will need to provide Dual with a few pieces of information about your app. We are required to include your developer information in our hardware product plan before your app will be approved by iTunes. *This is an Apple requirement, not a Dual requirement, and without your developer information added to the XGPS170 product plan, your app will be rejected by iTunes.*

Please provide us with the following information:

- The full name of your app as it appears (or will) in the App Store
- Your app version number (the one in the Info.plist file)
- Your app's bundle identifier. This is the unique CFBundleIdentifier that specifies your application, e.g. com.domain.app

Apple is *very* picky about this information: a misspelled or omitted word will cause a rejection. Please double check that the information you provide us is 100% correct. If you need to change it after you send it to us, please let us know before you submit the app.

Fortunately, you only need to provide this information to us once. We do not need to update the product plan with new version numbers, so you do not need to notify us prior to releasing updates to your app. You remain in complete control of your own release schedule.

However, when you submit *each* release of your app to iTunes for review, ***be sure to include this sentence in the Remarks section:*** “approved app under PPID #100314-0004 from Namsung Electronics.” This will help avoid unnecessary delays in the app review process.

When we add your information to our hardware product plan, please be aware that it can take several days for the change to be approved by Apple and propagate through their system. Much like the app submission process, there is very little information from Apple during the time when our hardware product plan is updated with your information and when it is approved. Furthermore, neither Dual nor you are automatically notified by Apple if/when the product plan change was approved or not. So please take this variable window of time into account when planning an app release. We recommend sending your developer/app information to us as early as possible so that we can enter it into the product plan and remove that obstacle for you.

### Other Boring but Important Things to Know

- The most challenging aspect to incorporating the XGPS170 into your app will be managing the app’s UI and workflow because the device can connect/disconnect without warning:
  - The user can (and will) turn the XGPS170 on or off at any point, and likely when your application is in the background.
  - Since the XGPS170 and the iOS are both mobile devices, the XGPS170 can go in and out of Bluetooth range at any point.
  - The Bluetooth range will decrease as the battery in the XGPS170 gets closer to being exhausted.

The good news, however, is that the `Blackjack` class handles the connect/disconnect process for you. Your code will just need to monitor whether the `Blackjack` is connected or not. The `Blackjack` class provides two Boolean variables which indicate the current connection status of the XGPS170, and notifications are also generated by the `Blackjack` class to inform your app of changes to the connection status. Please see the sample code for examples.

- The `Blackjack` class and its categories represent the entire API you will need use to interface to the device. It provides a robust connection to the device over a wide range of older (slower) and newer (faster) iOS devices. If you see method calls or other logic which appears to be redundant or counter intuitive, it probably exists for a reason – **test your code carefully and with multiple iOS devices if you make changes to the `Blackjack` class.**
- The `Blackjack` device provides ADS-B weather, traffic and GPS information in an output format that mimics the output of the Garmin GDL90. (This spec is available on the web from the FAA [here](#).) Since not all apps can parse this format, this API is available in two versions:
  - The “Lite” version simply manages the Bluetooth connection to the XGPS170 and provides an access point for your app to tap into the GDL90-formatted data stream from the XGPS170. Use this version of the SDK if your app can already parse GDL90-formatted data.

- The “Full” version of the API also manages the Bluetooth connection to the XGPS170, and will decode the ADS-B weather and traffic information into standard data types and objects.
- The `Blackjack` class is written using categories in order to make the code more manageable. When adding the API to your app, make sure you copy all of the files from the `Blackjack` folder in the sample app (Lite or Full) into your app to avoid compilation errors.
- **Important note if you are using the Lite version of the SDK:** the XGPS170 has the ability to send NMEA data as part of the data stream. This NMEA data is not formatted as a GDL90 sentence, but appears as NMEA-formatted plain text sentences. The NMEA data can vary in length but is sent as a single block inserted between GDL90 sentences in the stream, and is transmitted once per second. Your GDL90 parser will need to properly ignore this data.
  - The only app which enables this function is Dual’s Status Tool app, and it disables the function before the app exits. However, once the function is enabled, it stays enabled. Since the user may turn off the XGPS170 before closing the app, it is possible for your app to encounter NMEA sentences in the data stream when the device is turned back on again.
- Because GPS data can be obtained from the XGPS170 without using Core Location, your app can obtain position data without activating the GPS receiver in 3G/4G iOS devices. There are two significant benefits to this:
  - Your app can still receive GPS information when the iOS device is in Airplane Mode.
  - The battery in the iOS device will last significantly longer because the internal GPS chipset can significantly drain the iPhone/iPad battery.
- However, if Core Location is not being used, your app may do things differently than the user expects:
  - The top status bar on the iOS device screen will not display the GPS indicator.
  - The Location Services setting will have no effect on your app, and your app will not appear in the list of apps on the Location Services screen.

Be sure to read and understand Apple’s privacy guidelines about notifying the user when your app wants to use location information.

- Last but not least, you will benefit from some free advertising for your app by using this SDK. The first time the XGPS170 pairs with an iOS device, the iOS notifies the user that they need to install software from iTunes and asks the user if they want to go to the iTunes store. If the user taps “Yes”, the iOS opens the App Store automatically and shows a specific list of apps which support the XGPS170. This effectively means that new owners of the XGPS170 will automatically be shown your app on the iTunes store!

## About the XGPS170 Receiver

### Device Modes

The XGPS170 has two modes of operation, “GPS” & “ADS-B”, controlled by a slide switch on the side of the product:

- GPS mode activates just the GPS receiver.
- ADS-B mode activates both the GPS and ADS-B receivers.

There are also four LEDs on the XGPS170:

- Battery LED. Normally off. Flashing red when battery power is low. Red when charging. Green when charging is complete.
- Bluetooth LED. Flashing when looking for a device. Solid when at least one device is paired to the unit.
- GPS LED. Flashing green while acquiring a GPS signal. Solid green when locked on and a valid 2D or 3D position is available.
- ADS-B LED. Solid when in ADS-B mode. Pulses when ADS-B data is being received.
- The XGPS170 can connect to two Bluetooth devices simultaneously and delivers the same data to both devices. Data is also available simultaneously on the USB port.

The internal GPS will continue to track position for 10 minutes after apps stop requesting GPS information. After 10 minutes of idle time (no apps requesting GPS), the GPS receiver will go into sleep mode. It automatically wakes when GPS information is requested again.

The XGPS170 produces output compatible with iOS devices as well as Android, Windows, OS X and Linux devices. (The XGPS170 firmware currently will not support both iOS and non-iOS devices at the same time.) The output format is controlled by a second switch on the side of the device. The “Apple” position is for iOS devices. The “Normal” position is for all other devices, including OS X.

The XGPS170 uses the Bluetooth Serial Port Protocol (SPP) profile on the Bluetooth connection, and a generic USB CDC serial driver can be used on the USB connection. Communication happens via a bi-directional serial port connection at 9600 baud.

The output of the XGPS170 is formatted to mimic the output of a Garmin GDL-90. If your app can already parse GDL-90 formatted ADS-B+GPS data, very little additional integration is required to make the XGPS170 work with your app.

If your app does not parse GDL-90 formatted data, the API in this SDK can provide decoding of the encoded data.

### Position Information

Position data is available from the XGPS170 in three ways:

- Through the iOS CoreLocation library (both GPS and ADS-B modes)
- Through the Ownship messages in the GDL-90 formatted data stream (ADS-B mode only)
- Through individual variables in the Blackjack class (ADS-B mode only)

It is not a requirement to use CoreLocation in order to obtain GPS position information.

NOTE: The iOS will dynamically switch between GPS sources if more than one source is available. According to Apple, it is not possible to know which position source is currently in use, or to know when the iOS device switches sources. So if your app uses CoreLocation, you are not guaranteed to

be getting GPS information from the XGPS170 unless the iOS device is in Airplane Mode. (Alternatively, you can use the GPS information in the Ownship message.)

### **ADS-B Data Output**

ADS-B data is only available when the mode switch is set to ADS-B.

When an ADS-B signal is present, weather and TIS-B traffic data (if any) will be transmitted. If no ADS-B signal is available, only the heartbeat and ownship information is transmitted.

Air-to-air traffic messages will be transmitted whenever they are received.

The XGPS170 will transmit battery charge, charging status and GPS position information every second, even if an ADS-B signal is not present.

## About the XGPS170 SDK

This SDK contains these pieces:

- An Objective-C class, named *Blackjack*, which acts as the API you should use in your app
- Two sample app XCode projects
- Documentation (what you're reading now)
- A sample data file

### Lite and Full Versions of the Blackjack Class

This SDK actually contains two versions of the XGPS170 interface API: a Lite and a Full version.

The Lite version is intended for apps which can already parse GDL-90 formatted data streams. It manages the Bluetooth connection with the XGPS170 and provides an access point for your app to obtain data from the iOS Bluetooth buffer. The methods and data types associated with decoding GDL-90 formatted data have been stripped out of the Lite version of the Blackjack class in order to minimize the code size & memory requirements.

The Full version is a superset of the Lite version, providing the same Bluetooth connection management, plus the methods and data objects required for obtaining decoded ADS-B and GPS data. The Full version is intended for apps which cannot parse GDL-90 formatted data stream.

Neither version of the Blackjack class contains any UI objects. The Blackjack class is purely an interface and data management class.

### Sample Code

Two sample XCode projects are included with this SDK. *XGPS170\_Lite* implements the Lite version of the SDK. *XGPS170\_Full* implements the Full version of the SDK.

The *XGPS170\_Lite* sample app does very little that's visually interesting. It simply serves as an example of how to bolt the Blackjack class onto your app if your app can already parse GDL-90 formatted data.

The *XGPS170\_Full* sample app implements the Full version of the SDK showing device data along with displays of weather data. The sample app is meant to be an example of how to use access to the decoded ADS-B data through the Full version of the SDK.

### Testing

If you are in an area where there is no ADS-B reception on the ground, you're not out of luck. You have three options:

- Two kinds of sample data are included in the SDK:
  - The Full version of the SDK includes small snippets of ADS-B data in the file named `BlackjackTestData.h` which can be run via the `runTestData` method. See the comments in `BlackjackTestData.h` for a description of the data snippets.
  - A large block of data recorded during a 45 minute test flight is also included with the SDK. This file can be imported into the demo app in the Full SDK and the app will parse this file as if it were coming directly from the XGPS170. See the **Test Data** section below for specific instructions on how to do this.
- You can record your own data for analysis later. The XGPS170 produces the same output on both the Bluetooth and USB connections. So it is possible to simply record the output of the

XGPS170 using a PC, store the output to a file, and then import the file into your app. See the **Test Data** section below for specific instructions on how to do this.

- There is firmware available from Dual which makes the XGPS170 produce a looping stream of pre-recorded ADS-B data, just like it was receiving a live signal. Please contact us for more information.

More sample data is available upon request.

## Helpful Debugging Hints

- You can use the XCode debugger and console window while the XGPS170 is communicating with your app.
- If your app crashes, or you stop the execution of your code through the console/debugger, the communication stream with the XGPS170 may not close properly. If that happens, you may need to power cycle the XGPS170 before your app can communicate with the device again.
- If you need to reboot the XGPS170, make sure the iOS realizes that the connection with the device has terminated before powering the device back on:
  - Turn off the XGPS170.
  - Wait for the iOS to realize the device is no longer connected. (This can take several seconds, particularly on older iPod touch models.) Watch for the Bluetooth icon in the status bar on your iOS to turn gray.
  - Turn the XGPS170 back on, and wait for the Bluetooth light on the device to stop flashing.
- The iAP daemon in the iOS can become cantankerous after multiple occurrences of broken streams or a very large number of disconnect/reconnect cycles. If you start having difficulty with the Bluetooth pairing process, power down both the XGPS170 and the iOS device and restart both of them.

## Best Practices

Here are a few recommended best practices:

- Always notify the user when the XGPS170 connects or disconnects.
- When your app comes to the foreground, check to see whether or not the XGPS170 is connected or disconnected. The user could have shut off the device while your app was in the background. The **RefreshUIAfterAwakening** notification is generated for this purpose.
- It is highly recommended that you show XGPS170 device status information somewhere within your app, including battery level, connection status and whether your app is receiving information from the XGPS170 (like the Heartbeat message). This will greatly help when users contact you with “Why isn’t it working?” questions.
- The Blackjack class releases the connection to the XGPS170 when your app goes into the background, and we recommend that you don’t override this behavior. The connection is *not* automatically reopened when your app comes to the foreground again. Your app *must* check the connection status of the XGPS170 and start the flow of data again (see **Starting and Stopping Data Flow from the XGPS170** below).

## iOS Compatibility

The Blackjack class has been tested up to iOS 7.1.

The *iOS Deployment Target* in your XCode project's Build Settings should be set to 5.1 or above.

The Blackjack class assumes your project will use automatic reference counting (ARC).

## Protocols

The XGPS170 provides ADS-B and GPS data through an accessory protocol named `com.dualav.xgps170`. (It also supports the older `com.dualav.xgps150` protocol in GPS-only mode.) In order to access data from the device, you must enter this protocol name into your app's `Info.plist` file under *Supported External Accessory Protocols*:

▼ Supported external accessory protocols	Array	(1 item)
Item 0	String	<code>com.dualav.xgps170</code>

## Integration Checklist

After you've read through the Lite and/or Full sections below, use this checklist to double check you have completed everything necessary to work with the Blackjack API:

1. Lite SDK users will need to add the **ExternalAccessory** framework to your XCode project. Full SDK users will need **ExternalAccessory**, **QuartzCore**, **CoreText**, **CoreGraphics** and **CoreLocation**. The demo app also uses **MapKit**.
2. Add the **Blackjack** class and it's categories to your project:
  - a. Add all of the files in the *Blackjack* folder (Lite or Full version) into your project.
  - b. Import *Blackjack.h* into your project. You may also need to import *Blackjack+LifeCycleMgr.h*, *Blackjack+ADSBDDataMgr.h*, *Blackjack+ControlMgr.h* and *BlackjackDataStructures.h* depending upon the needs of your classes. (Compiler errors will make it obvious if you need to import additional header files.)
3. Instantiate the Blackjack object in your app delegate.
  - a. It is best to add the Blackjack object to your app delegate, and then access the device object in your classes via the delegate. Using the Blackjack object this way will ensure that the Bluetooth connection is maintained properly.
  - b. Initialize the Blackjack object in the delegate's `application:didLaunchWithOptions:` method:

```
self.blackjack = [[Blackjack alloc] init];
```
  - c. Add the Blackjack-related lifecycle methods to the each of the corresponding application lifecycle methods in your app delegate. This is perhaps best explained by example, so refer to the `AppDelegate.m` file in either of the XGPS170 sample apps. Copy the code from these five delegate methods into your app's delegate:
    - i. `applicationWillResignActive:`
    - ii. `applicationDidEnterBackground:`
    - iii. `applicationWillEnterForeground:`
    - iv. `applicationDidBecomeActive:`
    - v. `applicationWillTerminate:`
4. Register for the notifications you are interested in.
5. Compile for a target of iOS 5.1 or higher.
6. In your app's info.plist file, add a "**Supported External Accessory Protocol**" with a value of "**com.dualav.xgps170**".
7. Do something interested with the decoded ADS-B data. Access points for the decoded data are identified by **//TODO:** markers in `Blackjack.m` and `Blackjack+ADSBDDataMgr.h`. The pickoff point for raw data from the XGPS170 is in the `stream:handleEvent:` method in `Blackjack.m`.
8. If needed, implement the start/stop notifications as appropriate. (See below.)

## Using the Lite Version

The Lite version of the API provides no decoding of the incoming data. It simply manages the Bluetooth connection and provides a place for your app to obtain the streaming data.

The XGPS170 transmits heartbeat, ownship and ownship altitude messages once per second, even if no ADS-B signal is being received. Uplink and/or traffic messages (message ID 0x14) will be transmitted when received from the ADS-B ground station. Basic and Long Report messages are not transmitted unless first enabled. See the **XGPS170 Device Control** section for information on how to enable/disable these traffic messages.

## Detecting that the XGPS170 is Connected

There are two Boolean values in the Blackjack class which can be monitored to determine the connections status of the XGPS170: `isPaired` and `isConnected`. The value of `isPaired` is true when the XGPS170 is paired via Bluetooth, but not sending data. The value of `isConnected` is true when the XGPS170 is both paired and sending data.

## Starting and Stopping Data Flow from the XGPS170

GPS and ADS-B data flow is started automatically when the Blackjack class is initialized. However, data can be started (or stopped) at any time in your app.

To start data flowing, post a notification named `ADSBDData_Start`:

```
NSNotification *startNotification =
    [NSNotification notificationWithName:@"ADSBDData_Start" object:self];
[[NSNotificationCenter defaultCenter] postNotification:startNotification];
```

To manually stop data flowing from the device, post a notification named `ADSBDData_Stop` in a similar fashion.

You can also set the Blackjack class variable `appWantsADSBDData` to YES to start or NO to stop data flow, but this will not have an immediate effect.

When your app is moved into the background or is otherwise suspended, the Blackjack class stops the data flow from the device. **IMPORTANT NOTE:** You will need to restart data flow when your app resumes operation in the foreground.

## Obtaining Data from the XGPS170

As new data arrives into the iOS device, the iOS buffers the stream data and makes it available to your app. The place to tap into this stream is the `stream:handleEvent:` method in the `Blackjack.m` file. Look for a `//TODO:` comment at the location where you should insert the method call to your data handling routine.

## Device Specific Information

The XGPS170 provides device status information in a GDL90 sentence with the message ID 0x7a. The Lite version does not decode battery charge level & charging status, but this can be easily done in your app:

- Battery charge level is a 16 bit number, divided across the first two message bytes (MSB first) immediately following the message ID.
  - A charge value of 0x1004 or greater is 100% charge.
  - A charge value of 0x0DAC is 0%.

- The fifth byte after the message ID contains the charging status: a bitwise AND of byte 5 with 0x04 will give charge status. A non-zero result signifies that the battery is charging. A zero result indicates the battery is not charging.

NOTE: the battery charge level value is not valid when the device is charging.

## Using the Full Version

The Full version of the API provides objects and data types containing decoded position information, graphical and textual weather information, and traffic data. The API also decodes device status information, including battery level and charging status.

The decoded weather and traffic data is not stored by the SDK. It is up to your app to process and store the decoded data. The exception is graphical weather. Both regional NEXRAD and CONUS NEXRAD weather data is stored by the SDK so that an image can be obtained at any point.

However, the Blackjack class does not automatically clear out old weather data from memory. Incoming new data simply overwrites the old data. Methods are provided to clear the regional and CONUS weather data, and these should be called after your app has requested the imagery it needs.

## Detecting that the XGPS170 is Connected

There are two Boolean values in the Blackjack class which can be monitored to determine the connections status of the XGPS170: `isPaired` and `isConnected`. The value of `isPaired` is true when the XGPS170 is paired via Bluetooth, but not sending data. The value of `isConnected` is true when the XGPS170 is both paired and sending data.

## Starting & Stopping Data Flow

GPS and ADS-B data flow is started automatically when the Blackjack class is initialized. However, data can be started (or stopped) at any time in your app.

To start data flowing, post a notification named `ADSBDData_Start`:

```
NSNotification *startNotification =  
    [NSNotification notificationWithName:@"ADSBDData_Start" object:self];  
[[NSNotificationCenter defaultCenter] postNotification:startNotification];
```

To manually stop data flowing from the device, post a notification named `ADSBDData_Stop` in a similar fashion.

You can also set the Blackjack class variable `appWantsADSBDData` to YES to start or NO to stop data flow, but this will not have an immediate effect.

When your app is moved into the background or is otherwise suspended, the Blackjack class stops the data flow from the device. IMPORTANT NOTE: You will need to restart data flow when your app resumes operation in the foreground.

## Notification of Received Data

Data from the XGPS170 is transmitted to the iOS device through a series of predefined messages following the GDL-90 protocol, and a complete cycle of these messages occurs once per second. Because data can originate from ground towers and other aircraft, the information sent from the XGPS170 can vary each cycle.

At a minimum, the XGPS170 will send battery level, charging status, and GPS position information once per second. When the receiver is in range of ADS-B ground or air-to-air signals, weather and traffic information will also be included in the message cycle.

The recommended way for your app to know when and what new data is available is by listening for the `HeartbeatMessageReceived` notification and then checking the value of a variable named `adsbDataAvailable`. Treat `adsbDataAvailable` as a set of bit flags indicating what type of data

has arrived. Each time new data of specific type is received, the corresponding bit flag in `adsbDataAvailable` is set:

Variable	Type
<code>adsbDataAvailable</code>	<code>uint32_t</code>

Type of data received	Bit position	Binary Value
<reserved>	0	0000 0000 0000 0001
Traffic	1	0000 0000 0000 0010
NOTAM	2	0000 0000 0000 0100
METAR	3	0000 0000 0000 1000
TAF	4	0000 0000 0001 0000
D-ATIS	5	0000 0000 0010 0000
AIRMET	6	0000 0000 0100 0000
SIGMET	7	0000 0000 1000 0000
Special Use Airspace Update	8	0000 0001 0000 0000
PIREP	9	0000 0010 0000 0000
WINDS	10	0000 0100 0000 0000
TWIP	11	0000 1000 0000 0000
Regional NEXRAD	12	0001 0000 0000 0000
CONUS NEXRAD	13	0010 0000 0000 0000
<reserved>	14	0100 0000 0000 0000
<reserved>	15	1000 0000 0000 0000

For example, if the value of `adsbDataAvailable` is 4096, new regional NEXRAD data is available. If the value is 524, one or more PIREPs, METARs, and NOTAMs have been received.

Your code is responsible for clearing bits in `adsbDataAvailable`. For example, the sample app in the SDK clears `adsbDataAvailable` to zero every time a heartbeat message is received, since that indicates the start of a new data cycle from the XGPS170.

The Blackjack class provides several other notifications which your app can subscribe to. Some of the notifications are overlapping in their meaning, so you may not need to use all of them. **IMPORTANT NOTE:** Notifications are not generated if your app is backgrounded. See the **Notifications** section below.

### Using the Decoded Data

GPS position information and device data (battery level, charging status, etc.) are maintained by the Blackjack class and continually refreshed, so it is acceptable to simply check these values at any point.

However, you will need to add the decoded ADS-B data to your own data structures. The SDK decodes the incoming ADS-B data, but does not store it. The exception is that graphical weather data (NEXRAD and CONUS NEXRAD) are both decoded and stored.

You should access the decoded ADS-B data in `Blackjack+ADSBDDataMgr.m`. The code is marked with **//TODO:** comments in the locations where decoded data is available and ready to be added to your own structures.

## Graphical Weather

The SDK provides methods for retrieving the received weather data as georeferenced images suitable for display. If that is all you need, feel free to skip ahead to the **Displaying Graphical Weather** section.

If you are porting this code to another platform or want to render the raw data yourself, keep reading. Below is a brief explanation describing how the ADS-B system formats weather data and how the SDK organizes the decoded data.

### Broadcast Format of ADS-B Weather

The ADS-B system uses a block-and-bin system for dividing up the globe into regions for weather, instead of a more conventional TMS system. Additionally, the ADS-B system transmits groups of blocks together which, after decoding, look like wide strips instead of square tiles.

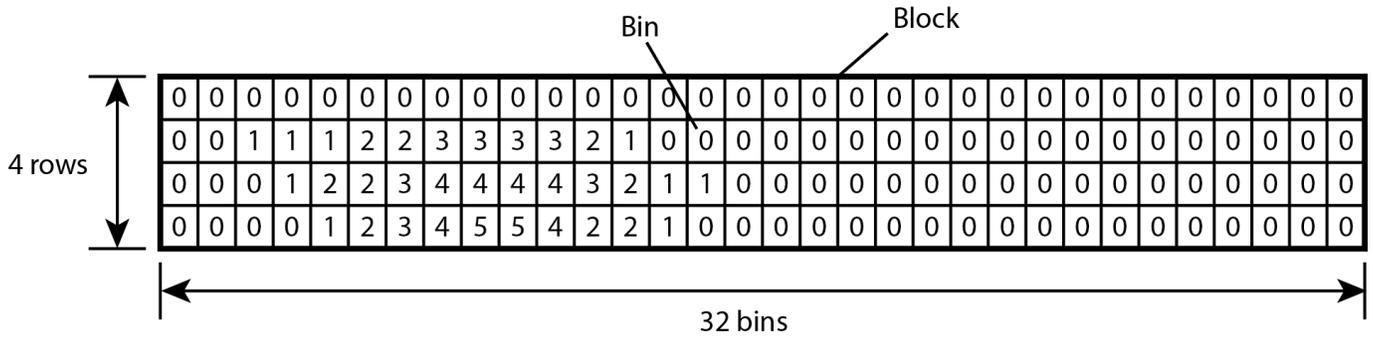
A block is made up of 4 rows of 32 bins each (see diagram below). Each bin is 1 minute of latitude “high” by 1.5 minutes of longitude “wide” (3.0 minutes wide above 60° latitude) and contains a value from 0-7 representing the weather intensity in that area. Each block, therefore, is 4 minutes of latitude high and 48 minutes of longitude wide (96 minutes above 60° latitude), and it contains 128 weather values. Blocks start at the intersection of the Prime Meridian and the Equator with the number 0, and increase to the right. (Blocks below the equator are negative, beginning with an oddly numbered block: negative 0).

CONUS and regional NEXRAD data blocks are transmitted separately, but utilize the same numbering system.

A common misconception is that the ADS-B weather picture shows cloud coverage. The reality is that ADS-B weather graphics only depict precipitation and its intensity.

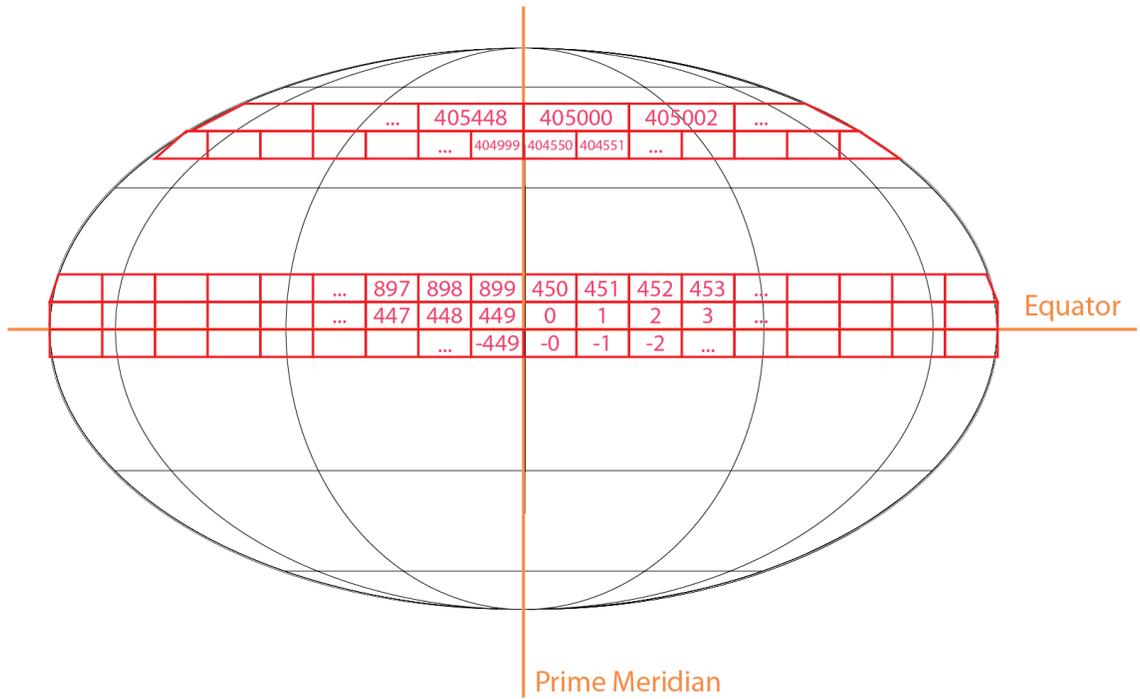
The resolution of NEXRAD radar varies with latitude but is roughly 1 square mile. Similarly, CONUS resolution is roughly 5 square miles.

NEXRAD weather is broadcast every 2 ½ minutes, although the data itself is updated every five minutes (said differently, the every other NEXRAD message is a redundant broadcast). CONUS NEXRAD weather is broadcast every 15 minutes, and each broadcast contains updated information. There are no redundant CONUS broadcasts.



1 bin = 1.5 min of longitude wide (or 3 min of longitude starting at 60° latitude),  
1 min of longitude high

1 block = 48 minutes wide (96 minutes above 60°)  
4 minutes high



When displaying the weather graphically, please note that the ADS-B specification states which colors should be used to represent the intensity numbers:

<b>Intensity Value</b>	<b>Displayed Color</b>
0	None
1	Green
2	Green
3	Amber or Yellow
4	Red
5	Red
6	Magenta
7	Magenta

### **Weather Data Structures in the Blackjack Class**

The Blackjack class automatically decodes incoming weather data, and stores it in memory in uncompressed form. A mutable dictionary is used to store the data. Each key in the dictionary is an NSNumber containing an ADS-B block number transmitted by the ground system, and the corresponding object is a NSData structure containing the intensity values for each bin in the block. The NSData objects are all 128 bytes each, and each byte will have a value from 0 to 7. The data order corresponds to the bin values, i.e. the first 32 bytes are the first row in the bin (starting at the upper left), the next 32 bytes are row two, etc.

There is one exception. An object for a key can contain a NSNull object if block was received from the ground network and does not have any precipitation, i.e. the block is all zeros.

If the dictionary returns nil for a block number, then no data has been transmitted from the ground system for that block number.

<b>If the Dictionary object is:</b>	<b>Then:</b>
NSData	There is a precipitation value greater than zero in at least one bin in the block.
NSNull	There is no precipitation in the block.
nil	The ground system has not broadcast anything for that block.

NEXRAD and CONUS weather data are managed in memory the same way, but occupy different dictionaries.

### **Displaying Graphical Weather**

**IMPORTANT NOTE:** The SDK does not automatically clear out old weather data. New data simply overwrites old data in the dictionary. If your app does not clear the old data before new data arrives, a “tail” of old weather data can form along the flight path. For example, if you are flying along and you receive a second NEXRAD update, the new data is simply added to the dictionary. Older blocks which are not contained in the newer broadcast will still exist in memory and will still be displayed. Once you have rendered the weather data into an image, you can clear the NEXRAD and CONUS arrays using the methods `clearNEXRADData` and `clearCONUSData`.

The SDK provides several methods for converting the received NEXRAD and CONUS weather into displayable images. Three methods will return images suitable for use with the iOS MapKit map view, and two methods return images which are not georeferenced to MapKit.

You can obtain an image of the received NEXRAD data using `createRegionalWeatherImage`:

Accessor Method	Return Type
<code>createRegionalWeatherImage</code>	<code>UIImage *</code>

You can obtain an image of the received CONUS data using `createNationalWeatherImage`:

Accessor Method	Return Type
<code>createNationalWeatherImage</code>	<code>UIImage *</code>

**IMPORTANT NOTE:** If you are using the iOS 7 paradigm of overlays for displaying images on a MapKit map view, you can easily overlay the returned images from these methods. Before generating the necessary overlay, use the convenience method `calculateNEXRADDataCoverageAreaUsingCONUS` to calculate the geographic region covered by the received data (NEXRAD or CONUS). You can then generate an overlay for that image which will properly georeference the image: see the `generateCurrentNEXRADWeatherImage` and `generateCurrentCONUSWeatherImage` methods in the demo app, as well as *NEXRADTestWeatherDataOverlay.m*, for usage examples. Also, note that you will need to discard and regenerate the overlays if the geographic area of the weather data changes. Otherwise, the data will be displayed in an incorrect location on the map.

You can also obtain an image of either NEXRAD or CONUS weather data for a specific MapKit region using `createWeatherImageWithRegion`:

Accessor Method	Return Type
<code>createWeatherImageWithRegion:(MKCoordinateRegion)coordinateRegion fromCONUSData:(BOOL)useCONUS</code>	<code>UIImage *</code>

You can obtain a non-georeferenced image using `createNEXRADImageWithUpperLeftLat`. This method allows you to specify a specific coordinate region and the size of the image (in pixels) which should be returned. The method can return an image of either NEXRAD or CONUS weather:

Accessor Method	Return Type
<code>createNEXRADImageWithUpperLeftLat:(double)upperLeftLat upperLeftLon:(double)upperLeftLon lowerRightLat:(double)lowerRightLat lowerRightLon:(double)lowerRightLon width:(uint32_t)width height:(uint32_t)height useCONUS:(BOOL)useCONUS</code>	<code>UIImage *</code>

A similar method allows you to specify the coordinate region using the upper left coordinates and a delta lat/lon:

Accessor Method	Return Type
<code>createNEXRADImageWithUpperLeftLat:(double)upperLeftLat upperLeftLon:(double)upperLeftLon deltaLat:(double)deltaLat deltaLon:(double)deltaLon width:(uint32_t)width height:(uint32_t)height</code>	<code>UIImage *</code>

useCONUS: (BOOL)useCONUS	
--------------------------	--

**IMPORTANT NOTE:** The ADS-B ground network broadcasts the weather payloads across multiple Uplink messages. Additionally, the weather messages may be interspersed with traffic or textual messages. So how do you know that there are no more weather messages yet to come? The ADS-B specification states that “end of broadcast” is indicated by a 10 second period elapsing without receiving a weather payload. So the demo app (not the SDK) implements a 10 second timer which restarts with each incoming weather message. The weather image data is generated only when the timer finally expires.

The SDK will allow your app to generate a weather image from current data at any time. Dispatch queues are used to put the image generation on a separate thread in order to minimize the CPU load.

The Blackjack class includes two NSDate variables holding the timestamp of the most recently received NEXRAD and CONUS message broadcasts: `timeOfLastNEXRADReception` and `timeOfLastCONUSNEXRADReception`.

#### *Alternative Options for Weather Data*

If none of the options above provides what you need, you can access the decoded weather data using other, lower-level methods in the SDK. These methods are not exposed in the header files, but can be used. Please contact us if you have any questions.

## Decoded Textual ADS-B Information

This list of items is broadcast by the ADS-B ground network as textual messages, and decoded by the SDK:

Text Product	Currently Broadcast*
NOTAMs	Yes
METARs	Yes
TAFs	Yes
D-ATIS	No
AIRMETs	Yes
SIGMETs	Yes
SUA Updates	Yes
PIREPs	Yes
Winds Aloft	Yes
TWIPS	No

\* At time of writing, not all types of information decoded by the SDK are actually being broadcast by the ADS-B ground network.

The SDK makes these decoded messages available as a `NSMutableDictionary`. One dictionary is generated for each type of message. For example, if `adsbDataAvailable` indicates that PIREPs and METARs have been received, the `Blackjack` class will have generated one dictionary containing PIREPs and another dictionary for METARs.

The key in each dictionary is the timestamp the message was decoded in the iOS device, and the corresponding object is the decoded message text. Several messages of the same type can be transmitted per cycle, so the dictionary will frequently contain multiple messages.

NOTAMs, in particular, will be a huge portion of the received data. A complete set of NOTAMs is broadcast every 10 minutes from the ground network, so you can expect to see multiple NOTAMs in the dictionary.

Keys are of type `NSDate` and objects are of type `NSString`. The dictionaries are available in `Blackjack+ADSBDDataMgr.m` file at locations marked by `//TODO:` comments.

## Traffic Data

The ADS-B system can transmit very detailed information about a traffic target, which can be either an aircraft or a ground vehicle. However, the ADS-B system permits two levels of detail for a traffic target, so not all targets are described with the same level of detail. In the GDL90 specification, these are called *Basic Reports* and *Long Reports*.

The GDL90 specification permits a third representation of traffic information called a *Traffic Report* (also called a Short traffic report in this documentation and the SDK code). This message contains abbreviated but adequate information about the traffic object. The XGPS170 ships from the factory configured to only produce the *Traffic Report* message. However, the SDK does allow you to turn *Basic* and *Long Reports* on or off independently of *Traffic Reports*. There is redundancy in the data between the *Basic/Long* and *Traffic* messages, so its unlikely an application will need both. Turning off the unnecessary message type will save Bluetooth bandwidth and extend the battery power in the XGPS170.

The Blackjack class uses two structs to describe traffic targets: `ShortTrafficObjectInfo` and `LongTrafficObjectInfo`. The `ShortTrafficObjectInfo` struct is populated with data from the *Traffic Report* message. `LongTrafficObjectInfo` structs are populated from *Basic Report* messages or *Long Report* messages. *Basic Reports* contain a subset of the information from *Long Reports*, so the elements in the `LongTrafficObjectInfo` structs pertaining only to *Long Reports* should be considered invalid when the struct is populated from a *Basic Report*. Both structs are defined in `BlackjackDataStructures.h` and the appendices contain a full description.

Some practical notes about traffic information from the ADS-B system:

- The ICAO Address value should be a unique value for each traffic target. However, we have seen cases in the real world where multiple targets on the ground are reporting the same self-assigned ICAO address.
- The ADS-B system can send multiple messages with non-redundant information for the same target. However, version 2.5 of the SDK does not combine messages for the same target.
- In practice, not all information is broadcast. For example, tail number and heading data are commonly unavailable.

The Blackjack class generates a `ShortTrafficMessageReceived` and a `BasicAndLongTrafficMessageReceived` notification when new traffic data has been received.

## GPS Information

The following position information is available in the Blackjack class:

	<b>Class Variable Name</b>	<b>Variable Type</b>	<b>Notes</b>
Valid/Invalid flag	gpsPositionIsValid	BOOL	True when GPS position is valid.
Latitude	Latitude	NSNumber	In degrees. Type: float
Longitude	Longitude	NSNumber	In degrees. Type: float
Altitude	Altitude	NSNumber	In feet. Type: int16_t
Velocity	Velocity	NSNumber	In knots. Type: uint16_t
Heading	Heading	NSNumber	In degrees (true). Type: float

All information is updated once per second. Altitude is reported in 25 foot increments per the published specification.

## XGPS170 Device Control

The XGPS170 has three characteristics which can be controlled by your app:

- LED brightness
- Which traffic messages to transmit to the iPad
- Whether or not to transmit detailed GPS information

### LED Brightness Settings

The brightness level of the four LEDs on the front of the XGPS170 is set using `setLEDBrightness:ADSBDimLevel:`

Accessor Method	Return Type
<code>setLEDBrightness:(uint8_t)brightLevel ADSBDimLevel:(uint8_t)dimLevel</code>	<code>void</code>

The `brightLevel` value controls the base brightness of the LEDs. Because the white ADS-B light on the XGPS170 blinks to indicate reception activity, the `dimLevel` value sets the lower brightness level for the ADS-B LED when it blinks. Values are in percent: 0 to 100. Recommended values for brightness settings are:

```
#define kDimmedLEDBrightnessLevel 5
#define kDimmedADSBBrightnessLevel 0
#define kNormalLEDBrightnessLevel 25
#define kNormalADSBBrightnessLevel 10
#define kBrightLEDBrightnessLevel 100
#define kBrightADSBBrightnessLevel 40
```

See the *SettingsPopover.h/.m* files for example usage.

### Traffic Message Settings

You can individually choose whether the XGPS170 transmits Short traffic messages, Basic and Long traffic messages, or both types of messages.

Short messages are turned on/off using `enableShortTrafficMessages` and `disableShortTrafficMessages`:

Accessor Method	Return Type
<code>enableShortTrafficMessages</code>	<code>void</code>
<code>disableShortTrafficMessages</code>	<code>void</code>

Basic and Long traffic messages are turned on/off using `enableBasicAndLongTrafficMessages` and `disableBasicAndLongTrafficMessages`:

Accessor Method	Return Type
<code>enableBasicAndLongTrafficMessages</code>	<code>void</code>
<code>disableBasicAndLongTrafficMessages</code>	<code>void</code>

See the *SettingsPopover.h/.m* files for example usage.

### Detailed GPS Data

If desired, two NMEA sentences containing detailed GPS information can be transmitted once every transmission cycle (i.e. between Heartbeat messages). PGSA and PGSV messages will be transmitted

when this option is enabled. The demo app uses these messages to determine whether the GPS position information is 2D or 3D, and whether WAAS data is in use.

Detailed GPS data is turned on/off using `enableDetailedSatData` and `disableDetailedSatData`:

Accessor Method	Return Type
<code>enableDetailedSatData</code>	<code>void</code>
<code>disableDetailedSatData</code>	<code>void</code>

See the *SettingsPopover.h/.m* files for example usage.

### IMPORTANT NOTES:

- The XGPS170 requires a finite amount of time to receive, acknowledge and apply changes in response to device control commands. Because of this time requirement, it is possible to send commands to the XGPS170 too quickly. To ensure commands do not get lost or stomped on, the Blackjack class implements a timed queue which accepts control commands from your code. The queue then releases commands to the device at an appropriate interval. It is strongly recommended that you use this queuing method in your app, and you can refer to examples in the *SettingsPopover.m* file. (We do break this “rule” for changing LED brightness settings because of the parameters which need to be sent to the method.)

The command queue is used by passing the accessor method name to `addCommandRequestToQueue`:

Accessor Method	Return Type
<code>addCommandRequestToQueue:(NSString *)methodToRun</code>	<code>void</code>

For example, to turn on Short traffic messages:

```
[blackjack addCommandRequestToQueue:
    NSStringFromSelector(@selector(enableShortTrafficMessages))];
```

- After issuing a change command, a query to confirm the change must be made before the same change command is sent again. Specifically, if you change the value of the traffic setting you must call:

```
[blackjack addCommandRequestToQueue:
    NSStringFromSelector(@selector(queryMessageFilterValue))];
```

- Similarly, if you change the value of the detailed GPS setting, you must make the same call:

```
[blackjack addCommandRequestToQueue:
    NSStringFromSelector(@selector(queryMessageFilterValue))];
```

## XGPS170 Device Status Information

The following device status information is available in the Blackjack class:

	<b>Class Variable Name</b>	<b>Variable Type</b>	<b>Notes</b>
Connection status	<code>isConnected</code>	BOOL	Your app should check this Boolean to see whether a valid communication session exists between the iOS and the XGPS170. If the value is YES, the device is available. If NO, the XGPS170 is off, out of range or disconnected.
Charging status	<code>isCharging</code>	BOOL	Indicates whether or not the XGPS170 is connected to power and charging. Updated once per second. Not valid unless ADS-B data is flowing.
Battery level	<code>batteryVoltage</code>	float	A value ranging from 0.0 to 1.0 indicating the charge level of the battery, e.g. 0.5 = 50% and 1.0 = 100%. Not valid unless ADS-B data is flowing. Also not valid while <code>isCharging</code> is true.
Battery level	<code>batteryIsLow</code>	BOOL	True when the battery level has dropped to a point where recharging is required very soon. Not valid unless ADS-B data is flowing. Also not valid while <code>isCharging</code> is true.
Device serial number	<code>serialNumber</code>	(NSString *)	A string containing the human readable model and serial number of the XGPS170. This can be treated as the unique ID of the device. Not valid until the device is paired to the iOS device.
Device firmware revision	<code>firmwareRev</code>	(NSString *)	A string containing the human readable firmware revision in the XGPS170. Not valid until the XGPS170 is paired to the iOS device.

## XGPS170 Device Status Information

The following device settings are available in the Blackjack class:

<b>Class Variable Name</b>	<b>Variable Type</b>	<b>Notes</b>
detailedGPSInfoEnabled	BOOL	If the value is YES, the PGSA and PGSV NMEA sentences are included in the stream data coming from the XGPS170. If NO, the these sentences are not being sent.
basicAndLongTrafficMessagesEnabled	BOOL	If YES, Basic and Long Traffic Messages are being transmitted by the XGPS170.
trafficMessagesEnabled	BOOL	If YES, Short Traffic Messages are being transmitted by the XGPS170.

## Notifications

The Blackjack class generates several different notifications, delivered through the default notification center:

- To know when the Blackjack connects or disconnects, register for the `BlackjackConnected` and `BlackjackDisconnected` notifications.
- To get regular “I’m alive & connected” updates from the XGPS170, register for `HeartbeatMessageReceived` notifications. This notification serves as a good general purpose indicator that the XGPS170 is alive and communicating.
- To be notified of position updates, register for `OwnshipMessageReceived` and/or `OwnshipAltitudeMessageReceived` notifications. The former is generated once per second when new lat/lon position information is sent from the XGPS170. The latter is generated when new altitude information is sent.
- To be notified of updates to device status information (battery level, charging status, etc.), register for `BlackjackDeviceDataUpdated` notifications.
- To be notified when new FIS-B or TIS-B data has been received from the ground transmitter network, register for `UplinkMessageReceived` notifications.
- To be notified when new traffic information is available, register for `ShortTrafficMessageReceived` and/or `BasicAndLongTrafficMessageReceived` notifications.
- When your app is brought to the foreground, the Blackjack class will generate a `RefreshUIAfterAwakening` notification after updating itself on the current status of the XGPS170 connection. The `isConnected` and `isPaired` Booleans will be valid after this notification is generated.

## Test Data

The file `BlackjackTestData.h` contains snippets of ADS-B broadcasts which can be used for test data samples. See the `runTestData` method at the end of *Blackjack+ADSBDATAMgr.m* for examples of how to use this test data.

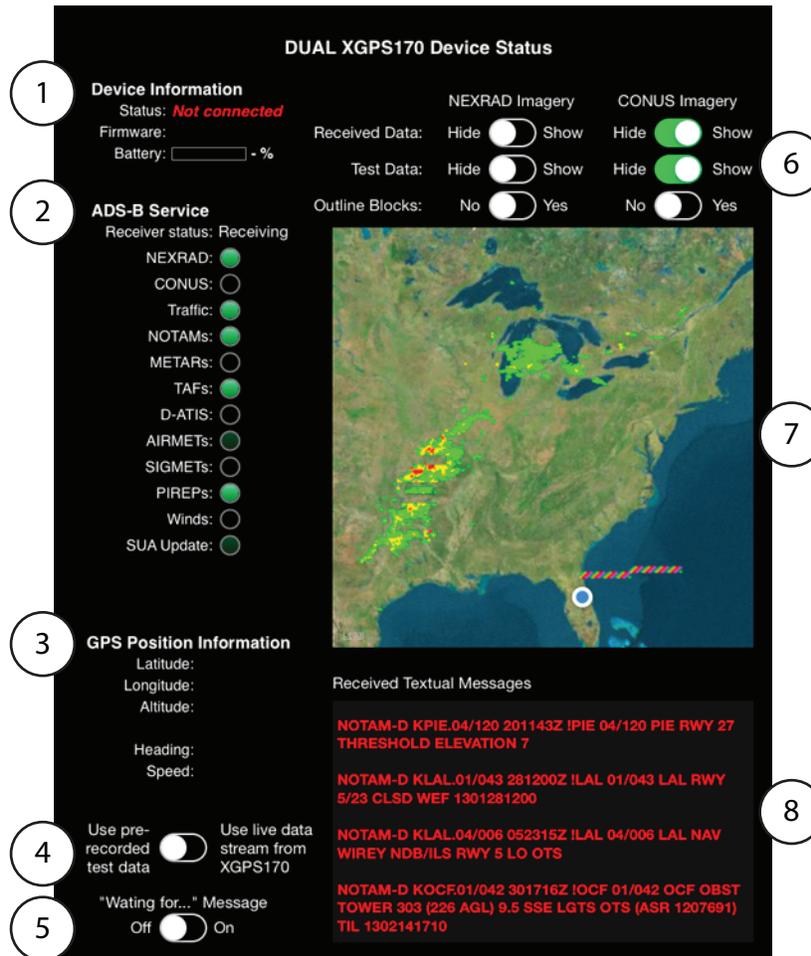
If you would like to import test data which you have recorded, or use the test flight data included with the SDK, do the following:

- Rename your sample data file to `ADSB_Test_Data.bin`
- Build and install the sample app on your iPad.
- Open iTunes, select your iPad, and click on the Apps tab.
- Scroll down to the File Sharing section, and locate “Demo App” in the list.
- Drag the data file to the Documents area. (No need to sync afterwards.)

Call the `runTestData` method to parse the data file.

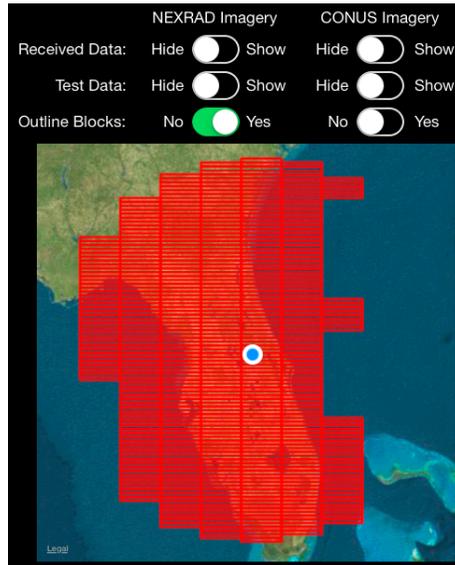
## Demo App

A screenshot from the demo app in the Full SDK is shown here:



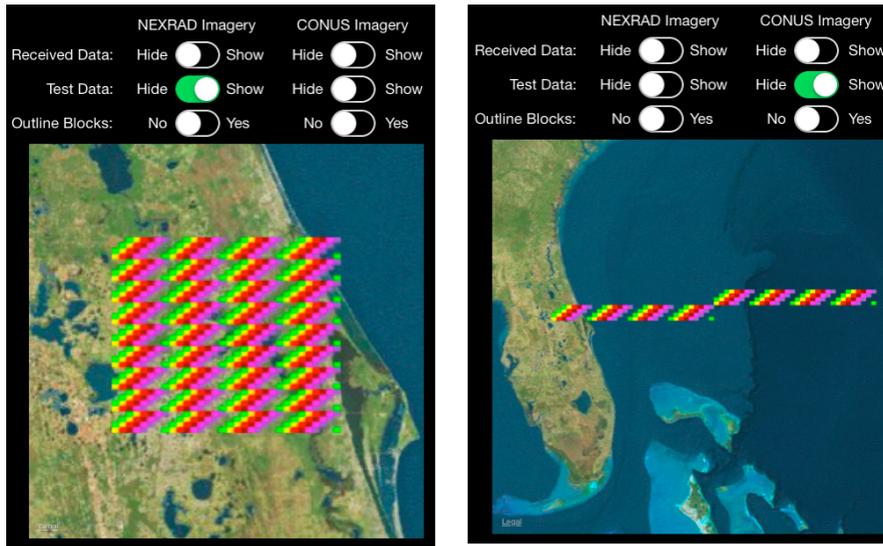
1. Basic device information is shown here.
2. As ADS-B data is received (or read in from a file), the type of information being processed will be indicated here.
3. GPS position information will show here when the iPad is connected to an XGPS170.
4. The switch changes the input data source between an externally connected XGPS170, and the internal test data. (You will need to install some test data before this switch does anything. See **Test Data** above.) Changing the position of this switch will clear the NEXRAD and CONUS images.
5. This switch hides/shows the "Waiting for the XGPS170" warning message which appears when no XGPS170 is connected.
6. These switches control which weather imagery is shown in the map view.
  - a) The Received Data switches turn on and off the NEXRAD and CONUS images generated from live or test weather data streams.
  - b) The Test Data switches will turn on or off the display of the synthetic test patterns. (NOTE: the synthetic test patterns are stored in the same memory location as regular weather, so showing the received data will also show the test data patterns unless the generateTestPatterns method in iPadViewController is disabled.)

- c) The Outline Blocks switches will show red rectangles around the individual FIS-B blocks received from the broadcast signal. This works best for debugging, and with small data sets.



- 7. This area shows NEXRAD and CONUS weather imagery. Traffic is not shown.
- 8. Decoded NOTAMs, METARs, TAFs, D-ATIS, AIRMETs, SIGMETs, PIREPs, TWIPs, Winds reports and special use airspaces are shown here in their raw form. This is a scrolling text box which self truncates to 9000 characters.

The test data will appear as colorful patterns in the NEXRAD and CONUS images over central Florida:



## License

The XGPS170 SDK is available to you under the BSD license:

Copyright (c) 2014 Dual Electronics Corp.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3) Neither the name of the Dual Electronics Corp. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**By using the code in this SDK you agree to accept the terms and conditions of the respective usage licenses.**

Trademarks and trade names mentioned are those of their respective owners.

## Appendix A: LongTrafficObjectInfo struct

Struct element	Data Type	Notes
hour	uint8_t	Zulu time the report was generated
min	uint8_t	(see above)
sec	uint8_t	(see above)
adsbTarget	bool	True if the traffic report originates from a ADS-B transmitter.
tisbTarget	bool	True if the traffic report originates from a TIS-B transmitter. <sup>1</sup>
surfaceVehicle	bool	True if the traffic report is not an aircraft.
adsbBeacon	bool	True if the traffic report originates from an ADS-B beacon.
adsrTarget	bool	True if the traffic report is generated from by ADS-Rebroadcast. <sup>1</sup>
icaoAddress	uint32_t	Unique ICAO address of the traffic target
icaoAddressIsSelfAssigned	bool	True if the ICAO Address is self-generated
latitude	float	
longitude	float	
altitudeType	bool	true = pressure altitude, false = geometric altitude
altitude	int32_t	In feet. Values less than -1000 = no altitude data
nic	float	In meters
airborne	bool	true = airborne, false = on the ground
supersonic	bool	true = supersonic, false = subsonic
northerlyVelocity	int16_t	in knots, only valid for airborne vehicles. Positive for northerly movement, negative for southerly movement.
northerlyVelocityAvail	bool	True when northerlyVelocity data is available from the broadcast source.
easterlyVelocity	int16_t	in knots, only valid for airborne vehicles Positive for easterly movement, negative for westerly movement
easterlyVelocityAvail	bool	True when easterlyVelocity data is available from the broadcast

		source.
groundSpeed	uint16_t	in knots, only valid for surface vehicles
groundSpeedAvail	bool	True when groundSpeed data is available from the broadcast source.
heading	float	See note 2.
trueTrackAngle	bool	true if heading value is a track angle
magneticHeading	bool	true if heading value is magnetic
trueHeading	bool	true if heading value is true
verticalVelocitySource	bool	true = barometric source, false = geometric source <sup>3</sup>
verticalVelocityDirection	bool	true = down, false = up <sup>3</sup>
verticalVelocityRate	int16_t	in feet per minute <sup>3</sup>
vehicleSizeIsValid	bool	<sup>3,4</sup>
length	uint8_t	length in meters <sup>3,4</sup>
width	float	width in meters <sup>3,4</sup>
gpsAntennaInfoIsValid	bool	
gpsAntennaOffsetAxis	bool	true = longitudinal antenna offset, false = lateral antenna offset
gpsAntennaOffsetSide	bool	valid for lateral offset only: true = right, false = left
gpsAntennaOffsetDistance	uint8_t	in meters, from the longitudinal (roll) axis of the aircraft or from the nose of the aircraft
adsbTransmittingSourceCoupledToUTCTime	bool	
uplinkFeedback	uint8_t	max number of successful uplink messages received
tisbSiteID	uint8_t	
isBasicMessage	bool	True means this is a Basic traffic message and the items below will be invalid. False indicates the following items will be valid (if available in the broadcast).
<b>Items below only available in Long messages</b>		
emitterCategoryDescr [38]	char	Emitter category is the description of the aircraft, e.g. large vortex heavy, paraglider, UAV, etc.
callSignFlightPlanID [9]	char	Tail number or flight plan ID
statusDescr [25]	char	Emergency status
mopsVersion	uint8_t	See note 5
sil	uint8_t	See note 5
transmitMSO	uint8_t	See note 5

sda	uint8_t	See note 5
nacP	uint8_t	See note 5
nacV	uint8_t	See note 5
nicBaro	bool	See note 5
capabilityCodes	uint8_t	See note 5
operationalModes	uint8_t	See note 5
csid	bool	See note 5
silSUPP	bool	See note 5
geoVertAcc	uint8_t	See note 5
saFlag	bool	See note 5
nicSUPP	bool	See note 5

Notes:

<sup>1</sup>There is one allowed case where an traffic target is either a TIS-B target or a ADS-R target, so both `tisbTarget` and `adsrTarget` can be TRUE.

<sup>2</sup>Invalid if `trueTrackAngle`, `magneticHeading` and `trueHeading` are all FALSE.

<sup>3</sup>For airborne targets, there is velocity information provided. Otherwise, size information is provided instead.

<sup>4</sup>For ground-based objects, size and GPS antenna offset may be available.

<sup>5</sup>See RTCA-282B for more information.

## Appendix B: ShortTrafficObjectInfo struct

Struct Element	Data Type	Notes																																													
status	unsigned char	Traffic alert status																																													
addressType	unsigned char	0 = ADS-B with ICAO address 1 = ADS-B with self-assigned address 2 = TIS-B with ICAO address 3 = TIS-B with track file ID 4 = surface vehicle 5 = ground station beacon 6-15 = reserved																																													
icaoAddress	uint32_t	Traffic's unique ICAO address																																													
latitude	float	Latitude of target																																													
longitude	float	Longitude of target																																													
altitude	int32_t	Altitude of target																																													
miscIndicators	unsigned char	<table border="0"> <tr> <td>bit3</td> <td>bit2</td> <td>bit1</td> <td>bit0</td> <td></td> </tr> <tr> <td>x</td> <td>x</td> <td>0</td> <td>0</td> <td>= trackHeading not valid</td> </tr> <tr> <td>x</td> <td>x</td> <td>0</td> <td>1</td> <td>= trackHeading is true track angle</td> </tr> <tr> <td>x</td> <td>x</td> <td>1</td> <td>0</td> <td>= trackHeading is magnetic heading</td> </tr> <tr> <td>x</td> <td>x</td> <td>1</td> <td>1</td> <td>= trackHeading is true heading</td> </tr> <tr> <td>x</td> <td>0</td> <td>x</td> <td>x</td> <td>= report is updated</td> </tr> <tr> <td>x</td> <td>1</td> <td>x</td> <td>x</td> <td>= report is extrapolated</td> </tr> <tr> <td>0</td> <td>x</td> <td>x</td> <td>x</td> <td>= on ground</td> </tr> <tr> <td>1</td> <td>x</td> <td>x</td> <td>x</td> <td>= airborne</td> </tr> </table>	bit3	bit2	bit1	bit0		x	x	0	0	= trackHeading not valid	x	x	0	1	= trackHeading is true track angle	x	x	1	0	= trackHeading is magnetic heading	x	x	1	1	= trackHeading is true heading	x	0	x	x	= report is updated	x	1	x	x	= report is extrapolated	0	x	x	x	= on ground	1	x	x	x	= airborne
bit3	bit2	bit1	bit0																																												
x	x	0	0	= trackHeading not valid																																											
x	x	0	1	= trackHeading is true track angle																																											
x	x	1	0	= trackHeading is magnetic heading																																											
x	x	1	1	= trackHeading is true heading																																											
x	0	x	x	= report is updated																																											
x	1	x	x	= report is extrapolated																																											
0	x	x	x	= on ground																																											
1	x	x	x	= airborne																																											
nic	unsigned char	navigation integrity category. (See GDL-90 spec pg. 21.)																																													
nacP	unsigned char	navigation accuracy category for position. (See GDL-90 spec pg. 21.)																																													
horizVelocity	uint16_t	horizontal velocity in knots																																													
vertVelocity	int16_t	vertical velocity in units of 64 fpm																																													
trackHeading	float	Target's heading																																													
emitterCategory	uint8_t	see GDL-90 spec page 23																																													
callsign[8]	char	Tail number or flight plan ID, if available.																																													
emergencyPriorityCode	unsigned char	emergency/priority code, if available.																																													