



MOTOROLA

Global Telecom Solution Sector

GPS Programming Guidelines

Document Overview	3
Disclaimer	3
Contact Information	4
Acronyms and Definitions	4
Location API	5
Overview	5
Accuracy	5
Motorola strives to achieve the highest possible accuracy; however, no GPS system can provide perfect accuracy in all situations. GPS accuracy can be affected by a multitude of potential error-introducing factors, including GPS satellite signal conditions and packet data availability. Position accuracy is not guaranteed nor implied.	5
Assist Data	5
Class Description	5
The Position API	5
ALMANAC Out Of Date	6
After permission is granted, the Java API brings up a system screen to prompt the user if the Almanac data in the phone is out of date or invalid, and the phone is not provisioned for packet data service. This is done only once after the phone powers up. If the user gives permission to override Almanac data, the Java API tries to retrieve position data. If user does not grant the Almanac override, the Java API returns the position with its attributes set to UNAVAILABLE and the status of <code>PositionConnection</code> set to <code>POSITION_RESPONSE_NO_ALMANAC_OVERRIDE</code> .	6
PositionConnection Class	6
• <code>delay=no</code> This option is designed to provide the serving cell latitude and longitude to an application immediately after it requests them. Because all other attributes in the <code>AggregatePosition</code> class may be set to UNAVAILABLE, an application should use this connection only to access the serving cell latitude and longitude. This request does not make use of the GPS chipset. If the handset is outside of the network coverage area, the serving cell latitude and longitude will be set to 0.	6
• <code>delay=low</code> This option provides a response to the application in a few seconds. New assist data is retrieved only if no assist data exists or if the assist data is older than the Maximum Assist Data Age (MADA). This operation is transparent to the application. This option is designed to provide all the position attributes with assistance from the Location Enhanced Service (LES) Server. To exercise this option, the device needs to have packet data service. The maximum response time for this type of request is carrier definable but is typically 32 seconds. If the API times out, the position will be returned with appropriate status and error code. If a low-delay request is made outside of the network coverage area, then the API will not get the assist data from the LES. The fix will proceed without assist data, and the timeout will remain at the low-delay value (e.g. 32 seconds).	7
• <code>delay=high</code> This option provides a response to the application where delay is longer than a <code>delay=low</code> setting. It provides for an assisted or autonomous fix for the application. The phone uses existing assist data only if it is available and valid; otherwise, the location fix shall proceed autonomously. The maximum response time for this type of request is carrier definable but is typically 180 seconds. If the response times out, position will be returned with the appropriate status and error code.	7
Method Descriptions	7
Using Position in Java	9
Code Examples	10
Recommendations	13

Document Overview

This guide describes the procedures used to develop a J2ME™ compliant application using Motorola's GPS API.

Detailed information on the Java 2 Micro Edition environment is not provided.

Disclaimer

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications can and do vary in different applications and actual performance may vary. Customer's technical experts must validate all "Typicals" for each customer application.

MOTOROLA MAKES NO WARRANTY WITH REGARD TO THE PRODUCTS OR SERVICES CONTAINED HEREIN. IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE GIVEN ONLY IF SPECIFICALLY REQUIRED BY APPLICABLE LAW. OTHERWISE, THEY ARE SPECIFICALLY EXCLUDED.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise.

No warranty is made that the software will meet your requirements or will work in combination with any hardware or applications software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

IN NO EVENT SHALL MOTOROLA BE LIABLE, WHETHER IN CONTRACT OR TORT (INCLUDING NEGLIGENCE) FOR ANY DAMAGES RESULTING FROM USE OF A PRODUCT OR SERVICE DESCRIBED HEREIN, OR FOR ANY INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, OR OTHER FINANCIAL LOSS ARISING OUT OF OR IN CONNECTION WITH THE ABILITY OR INABILITY TO USE THE PRODUCTS, TO THE FULL EXTENT THESE DAMAGES MAY BE DISCLAIMED BY LAW.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, so the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacture of the product or service.

Contact Information

Motorola, Inc.

IDEN Subscriber Group

Phone: 1-800-453-0920

URL: <https://commerce.motorola.com/idenonline/ideveloper/index.cfm>

Acronyms and Definitions

Table 0-1: Acronyms Used In This Guide

Acronym	Terminology	Definition
AGPS	Assisted Global Positioning System	Assisted Global Positioning System
API	Application Programming Interface	A set of classes, interfaces, and methods that can be used for creating an application.
CLDC	Connected Limited Device Configuration	Sun's J2ME™ configuration aimed at small, wirelessly connected, memory and user interface limited devices.
GPS	Global Positioning System	A navigational system using satellite signals to fix the location of a receiver on or above the earth's surface
JAD	Java Application Descriptor	Describes a J2ME™ application. Typically provides application size, distributor, version, etc.
WebJAL	Java Application Loader	Application used to load J2ME™ applications to the phone
JAR	Java Archive	Archive format used by J2ME™ applications for compression and packaging (similar to zip files).
J2ME™	Java 2 Micro Edition	The Java platform on the phone
MIDP	Mobile Information Device Profile	Sun's J2ME™ profile for connected, mobile devices
OEM	Original Equipment Manufacturer	Refers to the company or entity that manufactures the hardware described.
RMS	Record Management System	One of the mechanisms for persistent storage of data on the phone

Location API

Overview

Several iDEN phone models let users and developers access GPS position information such as latitude, longitude, altitude, speed, and etc. This feature is provided as a built-in application in the phone's standard ergonomics and as a J2ME™ API developers can use to create custom AGPS based applications. This section describes some of the phone's features that affect AGPS accuracy and availability from a J2ME™ MIDlet. This API provides access to NMEA stream of messages and can turn on the GPS chip set's NMEA capabilities.

Accuracy

The phone is designed to receive location fixes within a preset level of geographic accuracy as determined by the network provider. Using the Position API J2ME™ developers can retrieve a fix; however, the location value is not guaranteed to be within this level of accuracy. The API provides methods to determine whether a given fix is accurate or not.

Motorola strives to achieve the highest possible accuracy; however, no GPS system can provide perfect accuracy in all situations. GPS accuracy can be affected by a multitude of potential error-introducing factors, including GPS satellite signal conditions and packet data availability. Position accuracy is not guaranteed nor implied.

Assist Data

AGPS uses cellular assisted data to retrieve a location fix. The Position API provides J2ME™ developers with a method to determine whether cellular assisted data is used for a given fix.

Class Description

The API for the NMEA output messages is located in package `com.motorola.iden.position`

```
java.lang.Object
|
+ - com.motorola.iden.position.PositionConnection
```

The Position API

Position API is located in the `com.motorola.iden.position` package. The API provides the location functionalities required for Java applications to access GPS position information such as the following:

- Latitude
- Longitude
- Altitude
- Time Stamp
- Travel Direction
- Speed
- Altitude Uncertainty
- Speed Uncertainty

The Position API uses the GPS Privacy setting in the Main Menu of the phone when a MIDlet invokes the API. Based on the GPS Privacy setting value, the MIDlet does or does not have the access to the position information. The API will use the user's Privacy setting accordingly before providing position information. Some examples include:

- If the user's GPS Privacy setting is set to "Restricted", the Java API will return the position with all the attributes set to `UNAVAILABLE` and with the `PositionConnection`'s status code set to `POSITION_RESPONSE_RESTRICTED`.
- If the user's GPS Privacy setting is set to "Unrestricted", the Java API will be able to access GPS data and will return the position.
- If the user's GPS Privacy setting is set to "By Permission", the application is suspended, as the Java API brings up a system screen to prompt the user for permission to grant position access for this application. If the user does not grant permission, the Java API will return the position with all the attributes set to `UNAVAILABLE` and with the `PositionConnection`'s status code set to `POSITION_RESPONSE_RESTRICTED`. After selecting one of the permission options, the user needs to resume the application.

ALMANAC Out Of Date

After permission is granted, the Java API brings up a system screen to prompt the user if the Almanac data in the phone is out of date or invalid, and the phone is not provisioned for packet data service. This is done only once after the phone powers up. If the user gives permission to override Almanac data, the Java API tries to retrieve position data. If user does not grant the Almanac override, the Java API returns the position with its attributes set to `UNAVAILABLE` and the status of `PositionConnection` set to `POSITION_RESPONSE_NO_ALMANAC_OVERRIDE`.

PositionConnection Class

This interface supports the creation of a connection to the GPS receiver (driver). GPS position can be retrieved and status can be obtained after creating a connection. Only one connection is allowed at a time. This API must be called from a separate thread from the main application thread.

To get a `PositionConnection`, the MIDlet must use the generic `Connector` class. For example:

```
com.motorola.iden.PositionConnection sc =  
    (com.motorola.iden.PositionConnection)Connector.open(String name);
```

String name should be one of the following:

- name = "mposition:delay=no"
- name = "mposition:delay=low"
- name = "mposition:delay=high"

The following descriptions of delay values are based on the default settings. These settings are carrier definable and can differ among carriers. Java has no access to change these values.

- `delay=no` This option is designed to provide the serving cell latitude and longitude to an application immediately after it requests them. Because all other attributes in the `AggregatePosition` class may be set to `UNAVAILABLE`, an application should use this connection only to access the serving cell latitude and longitude. This request does not make use of the GPS chipset. If the handset is outside of the network coverage area, the serving cell latitude and longitude will be set to 0.

- `delay=low` This option provides a response to the application in a few seconds. New assist data is retrieved only if no assist data exists or if the assist data is older than the Maximum Assist Data Age (MADA). This operation is transparent to the application. This option is designed to provide all the position attributes with assistance from the Location Enhanced Service (LES) Server. To exercise this option, the device needs to have packet data service. The maximum response time for this type of request is carrier definable but is typically 32 seconds. If the API times out, the position will be returned with appropriate status and error code. If a low-delay request is made outside of the network coverage area, then the API will not get the assist data from the LES. The fix will proceed without assist data, and the timeout will remain at the low-delay value (e.g. 32 seconds).
- `delay=high` This option provides a response to the application where delay is longer than a `delay=low` setting. It provides for an assisted or autonomous fix for the application. The phone uses existing assist data only if it is available and valid; otherwise, the location fix shall proceed autonomously. The maximum response time for this type of request is carrier definable but is typically 180 seconds. If the response times out, position will be returned with the appropriate status and error code.

Only one request of `getPosition()` can be made or be pending at any time. If the application makes multiple requests without getting a response to the previous request, a null position value is returned or an exception is thrown. The next section provides more detail on this method.

Method Descriptions

`getPosition()`

Returns a position by using the same delay setting used for `Connector.open()`.

`getPosition(String name)`

Returns a new position with different delay parameters. This method also allows an application to obtain a fix with an accurate velocity and heading direction. Note that obtaining an accurate velocity and heading direction may cause a significant delay with weak GPS signal strength. In strong GPS signal coverage this operation may take no longer than a standard fix.

The argument required for accurate velocity and heading direction is as follows:

```
String name = "delay=low;fix=extended"; // or
String name = "delay=high;fix=extended";
```

`getPosition()` and `getPosition(String name)` are synchronous, blocking methods which means these methods block until a response, error, or timeout occurs. Closing the `PositionConnection` from a separate thread can unblock these calls. Once the connection is closed, it needs to be opened again using `Connector.open()`.

If the `PositionConnection` is closed while a `getPosition()` call is pending or a second call has been made to `getPosition()`, then `getPosition()` and `getPosition(String name)` return a null position. Unknown errors may occur during a location fix, which may also cause null position value to be returned.

`requestPending()`

Lets an application check for pending position requests on a connection before making a new request from another thread.

getStatus()

Returns the connection status response of the last location fix operation. This method should be called only after calling `getPosition()` or `getPosition(String name)`. The obtained position information should be used only when `getStatus()` returns `POSITION_RESPONSE_OK` status code. The following is a list of returned responses for this method:

- `POSITION_NO_RESPONSE` indicates that the device is not responding. No position information will be available, and all the attributes of the position will be set to `UNAVAILABLE`.
- `POSITION_RESPONSE_ERROR` indicates that an error occurred while retrieving the position. If possible, the serving cell latitude and longitude will be available, but all position's attributes will be set to `UNAVAILABLE`.
- `POSITION_RESPONSE_OK` indicates that the obtained position is a valid position. All position's attributes will be available.
- `POSITION_RESPONSE_RESTRICTED` indicates that the user has set the device to not provide the position information. No position information will be available, and the position's attributes will be set to `UNAVAILABLE`.
- `POSITION_WAITING_RESPONSE` indicates that the API is waiting for a response from the position device. `POSITION_WAITING_RESPONSE` will be returned if `getStatus()` method is called before `getPosition()` method.
- `POSITION_RESPONSE_NO_ALMANAC_OVERRIDE` indicates that the Almanac is outdated, and the user is restricted to override. No position information will be available, and all the attributes of the position will be set to `UNAVAILABLE`.

getNMEASentence(int type) *Available on i730 only*

Returns an NMEA Sentence for the specified type.

Following are the valid NMEA message types.

- `PositionDevice.GPGGA`
- `PositionDevice.GPGLL`
- `PositionDevice.GPGSA`
- `PositionDevice.GPGSV1`
- `PositionDevice.GPGSV2`
- `PositionDevice.GPGSV3`
- `PositionDevice.GPRMC`
- `PositionDevice.GPVTG`

If the message type is other than above, this method returns `IllegalArgumentException` will be thrown.

If the method cannot fulfill the request for an NMEA sentence, this method returns a null string .

This first time you call this message, it turns on the GPS chip for NMEA messages, so it's the application's responsibility to stop the NMEA request once it is done using it,

stopNMEASentence *Available on i730 only*

This method stops the NMEA request and turns off the GPS chip after 10 seconds. Call to this API only stops NMEA access and keeps connection open so application can use the same connection to retrieve the position fix or reuse it for NMEA messages.

Using Position in Java

getResponseCode()

Each position response has an associated response code. The `getResponseCode()` method in the `AggregatePosition` Interface allows an application to get the response code. The Application can use this response code to validate the position. The following is a list of returned response codes:

- `POSITION_OK` indicates that the obtained position is valid and accurate.
- `ACC_NOT_ATTAIN_ASSIST_DATA_UNAV` indicates that the location fix has timed out. The fix could not be accurately obtained since assistance data was not available.
- `ALMANAC_OUT_OF_DATE` indicates that the Almanac is out of date.
- `ACCURACY_NOT_ATTAINABLE` indicates that the location fix has timed out, and the requested accuracy is not attainable.
- `BATTERY_TOO_LOW` indicates that the battery is too weak to retrieve a fix.
- `FIX_NOT_ATTAIN_ASSIST_DATA_UNAV` indicates that the location fix has timed out because a fix is not attainable, and assist data is unavailable.
- `FIX_NOT_ATTAINABLE` indicates that the location fix has timed out because a fix is not attainable.
- `GPS_CHIPSET_MALFUNCTION` indicates that the GPS chipset is malfunctioning.
- `UNAVAILABLE` indicates that an unknown error has occurred. This is the default response code.

These response codes are used in conjunction with `PositionConnection.getStatus()` to determine the quality of the retrieved position. These values are only valid when either `POSITION_RESPONSE_ERROR` or `POSITION_RESPONSE_OK` have been returned.

The following table shows possible response codes for these two values:

PositionConnection Status Values	Response Codes
<code>POSITION_RESPONSE_OK</code>	<code>POSITION_OK</code> <code>ACCURACY_NOT_ATTAINABLE</code> <code>ACC_NOT_ATTAIN_ASSIST_DATA_UNAV</code>
<code>POSITION_RESPONSE_ERROR</code>	<code>FIX_NOT_ATTAINABLE</code> <code>FIX_NOT_ATTAIN_ASSIST_DATA_UNAV</code> <code>BATTERY_TOO_LOW</code> <code>GPS_CHIPSET_MALFUNCTION</code> <code>ALMANAC_OUT_OF_DATE, UNAVAILABLE</code>

getAssistanceUsed()

Checks if a fix has been retrieved using assistance.

Code Examples

```

void getViaPositionConnection() throws IOException {
    com.motorola.iden.PositionConnection c = null;
    String name = "mposition:delay=low";
    try{
        c = (PositionConnection)Connector.open(name);
        AggregatePosition oap = c.getPosition();
        // Returns the AggregatePosition which contains the position using the
        // parameter passed when connection was opened.
        // Application should only check status by calling getStatus() after
        // getPosition() or getPosition(String name) returns,
        // otherwise it returns the same status and is
        // considered an invalid call of getStatus().
        // check the status code for permission and almanac override
        if(c.getStatus() ==
com.motorola.iden.position.PositionConnection.POSITION_RESPONSE_RESTRICTED
        {
            // means user has restricted permission to get position
        }
        else if(c.getStatus() ==
com.motorola.iden.position.PositionConnection.
POSITION_RESPONSE_NO_ALMANAC_OVERRIDE)
        {
            // means device has Almanac out of date and User has not granted to
override
        }
        else if(c.getStatus() ==
com.motorola.iden.position.PositionConnection. POSITION_NO_RESPONSE)
        {
            // means no response from device
        }
        if (oap != null ) {
            if(c.getStatus() ==
com.motorola.iden.position.PositionConnection.POSITION_RESPONSE_OK)
            {
                // Good position
                // Check for any error from device on position
                // Application needs to check for null position
                if(oap.getResponseCode() == PositionDevice.POSITION_OK) {
                    // no error in the position
                    if(oap.hasLatLon()) {
                        // int value of Latitude and Longitude of the position in arc
                        // minutes multiplied by 100,000 to maintain accuracy or
                        // UNAVAILABLE if not available
                        int lat = oap.getLatitude();
                        int lon = oap.getLongitude();
                        // String representation of the Latitude and Longitude.
                        String LATDEGREES = oap.getLatitude(Position2D.DEGREES);
                        String LONGDEGREES = oap.getLongitude(Position2D.DEGREES);
                    }
                    if(oap.hasSpeedUncertainty()) {
                        // speed and heading value are valid
                        int speed = oap.getSpeed();
                        if (hasTravelDirection()) {
                            // heading is available
                            int travelDirection = oap.getTravelDirection();

```

```

    }
  }
  if(oap.hasAltitudeUncertainty()) {
    int alt = oap.getAltitude(); //altitude of position in meters.
  }
}
// handle the errors...or request again for good position
// or display message to the user.
else if(oap.getResponseCode() ==
  PositionDevice.ACCURACY_NOT_ATTAINABLE) {
  // the position information was provided but enough accuracy
  // may not be attainable
}
else if(oap.getResponseCode() ==
  PositionDevice.ACC_NOT_ATTAIN_ASSIST_DATA_UNAV) {
  // the position information was provided but enough accuracy
  // assistant data unavailable
}
} // end of position response ok
else if(c.getStatus() ==
com.motorola.iden.position.PositionConnection.POSITION_RESPONSE_ERROR)
{
  // indicate an error occurred while getting the position
  if(oap.getResponseCode() == PositionDevice.FIX_NOT_ATTAINABLE) {
    // means position information not provided (timeout)
  }
  else if(oap.getResponseCode() ==
    PositionDevice.FIX_NOT_ATTAIN_ASSIST_DATA_UNAV) {
    // means position information not provided (timeout) and
    // assistant data unavailable
  }
  else if(oap.getResponseCode() == PositionDevice.BATTERY_TOO_LOW) {
    // means battery is too low to provide fix
  }
  else if(oap.getResponseCode() ==
    PositionDevice.GPS_CHIPSET_MALFUNCTION) {
    // means GPS chipset malfunction
  }
  else if(oap.getResponseCode() == PositionDevice.ALMANAC_OUT_OF_DATE) {
    // means almanac out of date to get fix
    // This scenario occurs when user overrides almanac but device is
    // not packet data provisioned
  }
  else{
    // Unknown error occurs
  }
} // end of position response error
// position is null
} finally {
  if ( c != null)
  c.close();
}

```

New positions can be obtained using the following method on the same PositionConnection object until the close method is called.

```
AggregatePosition cell = c.getPosition("delay=no");
```

Or

```
AggregatePosition oap = c.getPosition("delay=low");
```

Or

```
AggregatePosition oap = c.getPosition("delay=high");
```

In addition, to obtain better accurate speed and direction

```
AggregatePosition oap = c.getPosition("delay=low;fix=extended");
```

Or

```
AggregatePosition oap = c.getPosition("delay=high;fix=extended");
```

The following is the code example of NMEA use:

```
try
{
    PositionConnection posCon =
(PositionConnection)Connector.open("mposition:delay=low");

    String temp1 = posCon.getNMEASentence(PositionDevice.GPGGA);
if(posCon.getStatus() == POSITION_RESPONSE_OK)
{
    if(temp1 != null && temp1.equals(""))
    {
        // valid GPGGA string, parse it to extract the required information
    }
    else if(posCon.getStatus() == POSITION_RESPONSE_RESTRICTED)
    {
        // User has not granted permission to access its location informatio
    }
    else if (posCon.getStatus() == POSITION_RESPONSE_NO_ALMANAC_OVERRIDE)
    {
        // User has not granted permission to override its almanac
information
    }
    else
    {
        // unusual error occurred
    }
}
catch(IllegalArgumentException ie) {
}
catch(Exception ex) {
}
}

n oap = c.getPosition("delay=high;fix=extended");
```

Recommendations

- The GPS receiver requires access to both the iDEN network and GPS satellite signals to obtain rapid fixes. It is recommended that once the first fix is obtained the application monitors the response codes and vary the times between position requests accordingly. This recommendation is to handle the real world case where an app requests fixes rapidly (less than 10 seconds apart) and then loses network and GPS coverage (by entering a parking structure, basement, etc.) The GPS system will continue to try to find the unit's position and will go into a longer integration or acquisition mode that, once started, may take so long to finish that it may miss GPS signals once back in coverage. The recommended practice is to make fixes rapidly until a response code of `FIX_NOT_ATTAINABLE` or `FIX_NOT_ATTAIN_ASSIST_DATA_UNAV` is returned several times in a row (for about 10 requests for `delay=low` and about 5 requests for `delay = high`). Once this scenario occurs, then the app may wish to start the acquisition over from the beginning in anticipation that the phone might be back in GPS coverage. To do so, the application must wait 15 to 20 seconds after receiving the last response code before requesting a new fix. After this pause, the application can continue requesting fixes at its normal frequency.
- The GPS Subsystem requires about one second to calculate a new fix, so any request for a new fix during this one-second period may result in the exact same position information including the time stamp. Therefore it is recommended that an application request a new position no more than once per second.
- If an application needs continuous position it is recommended to use “`delay=low`” once and “`delay=high`” there after even if the first fix does not succeed. The reason for this is because of network failures. When there is a network failure, there is a 12 to 24 second communication timeout from the LES.
- Use “`delay=no`” if the application only needs the serving cell latitude and longitude.
- Applications must handle all response codes returned by the `AggregatePosition.getResponseCode()` method and the `PositionConnection.getStatus()` method. `getStatus()` provides the connection's status after the fix and user interaction status with regards to permission. `getResponseCode()` provides information about the position itself.
- Applications must always check the speed uncertainty value before using speed and heading. Although it is counter-intuitive, the presence of speed uncertainty denotes that the speed and heading value are accurate. Therefore, if a call to `hasSpeedUncertainty()` returns true, the speed and heading returned by the API are valid.
- If an application calls `getPosition(String name)` method with the “`fix=extended`” tag, this method will return accurate velocity and heading direction; however, there is a time penalty since it takes longer to calculate the accurate velocity and heading direction when the method is called.
- The method, `PositionConnetion.getStatus()` provides the status of the connection when the method `PositionConnection.getPosition()` was called. Whereas, `AggregatePosition.getResponseCode()` returns the detailed response code.

- Getting a position for the first time after the phone powers on is referred to as a “cold start”. A position retrieved within ten seconds of the previous fix is referred to as a “hot start”. A position retrieved after ten seconds of the previous fix is a “warm start”. After 1 hour since the last fix will set the device back to “cold start”. Therefore, “hot start” is the quickest way of retrieving a fix.
- For i730 it is highly recommended that antenna is extended all the time while getting fixes
- There is a battery impact when the NMEA API is used heavily.
- If the application will need NMEA data again in less than 10 seconds, there is no value to call `stopNMEASentence()` because the GPS chip will stay on for 10 seconds after calling `stopNMEASentence()`.
- First call of `getNMEASentence()` will turn on the GPS chip and it stays on until application calls `stopNMEASentence()`.

MOTOROLA, the Stylized M Logo and all other trademarks indicated as such herein are trademarks of Motorola, Inc. ® Reg. U.S. Pat. & Tm. Off. © 2003 Motorola, Inc. All rights reserved.

Microsoft and, Microsoft WEB Explorer, are registered trademarks of Microsoft Corporation.

Java and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All other product or service names mentioned in this manual are the property of their respective trademark owners.

Software Copyright Notice

The Motorola products described in this manual may include copyrighted Motorola and third party software stored in semiconductor memories or other media. Laws in the United States and other countries preserve for Motorola and third party software providers certain exclusive rights for copyrighted software, such as the exclusive rights to distribute or reproduce the copyrighted software. Accordingly, any copyrighted software contained in the Motorola products may not be modified, reverse-engineered, distributed, or reproduced in any manner to the extent allowed by law. Furthermore, the purchase of the Motorola products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents, or patent applications of Motorola or any third party software provider, except for the normal, non-exclusive, royalty-free license to use that arises by operation of law in the sale of a product.