Location API

for Java™ 2 Micro Edition

Version 1.1

Java Community Process / JSR 179 Expert Group
<jsr-179-comments@jcp.org>

Location API: for Java™ 2 Micro Edition

Version	Published	Description
1.1	2009-03-13	Final version

Status: Maintenance Release 2 / Final Release 3

Released: 2009-03-13

RESEARCH AND EVALUATION LICENSE

NOKIA CORPORATION IS WILLING TO LICENSE THIS SPECIFICATION TO YOU ONLY UPON THE TERMS CONTAINED IN THIS LICENSE ("LICENSE"). PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE CAREFULLY. BY ACCESSING OR USING THE SPECIFICATION YOU WILL BE BOUND BY THE TERMS OF THIS LICENSE.

JSR 179 Location API for Java™ 2 Micro Edition ("Specification")

Specification Lead: Nokia Corporation ("Specification Lead")

Version: 1.1

Status: Maintenance Release 2 / Final Release 3

Release: 2009-03-13

1. NOTICE; LIMITED LICENSE GRANTS

1.1 The Specification Lead hereby grants You a non-exclusive, non-transferable, worldwide, royalty-free, fully paid-up, limited license (without the right to sublicense) solely under intellectual property rights licensable by the Specification Lead to analyze and to use the Specification for research, evaluation, optimization and development purposes. In addition You may make a reasonable number of verbatim copies of this Specification in its entirety for Your private or internal use, as applicable, in accordance with the terms and conditions of this License.

1.2 No rights are granted under this License for internal deployment, the creation and/or distribution of implementations of the Specification for direct or indirect (including strategic) gain or advantage, the modification of the Specification (other than to the extent of Your fair use rights) or the distribution of the Specification or making the Specification available for 3rd parties.

1.3 Except as expressly set forth in this license, You acquire no right, title or interest in or to Specification or any other intellectual property licensable by the Specification Lead and no other rights are granted by implication, estoppel or otherwise. The Specification may only be used in accordance with the license terms set forth herein. This License will terminate immediately without notice from Specification Lead if You fail to comply with any provision of this License.

2. TRADEMARKS

2.1 Nokia is a registered trademark of Nokia Corporation. Nokia Corporation's product names are either trademarks or registered trademarks of Nokia Corporation. Your access to this Specification should not be construed as granting, by implication, estoppel or otherwise, any license or right to use any marks appearing in the Specification without the prior written consent of Nokia Corporation or Nokia's licensors. No right, title, or interest in or to any trademarks, service marks, or trade names of any third parties, is granted hereunder.

2.2 You shall not be allowed to remove any of the copyright statements or disclaimers or other proprietary notices contained in the Specification and You are obliged to include the copyright statement and the disclaimers, if any, in any copies of the Specification You make.

3. DISCLAIMER OF WARRANTIES

3.1 SUBJECT TO ANY STATUTORY WARRANTIES OR CONDITIONS WHICH CAN NOT BE EXCLUDED, THE SPECIFICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OR CONDITION OF ANY KIND EITHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, AND STATUTORY ARE HEREBY DISCLAIMED. THE ENTIRE RISK ARISING OUT OF OR RELATING TO THE USE OR PERFORMANCE OF THE SPECIFICATION REMAINS WITH YOU.

3.2 THE SPECIFICATION MAY INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SPECIFICATION LEAD MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

4. LIMITATION OF LIABILITY

4.1 TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL THE SPECIFICATION LEAD OR ITS SUPPLIERS BE LIABLE FOR ANY LOST PROFITS, LOST SAVINGS, LOST REVENUE, LOST DATA, PROCUREMENT OF SUBSTITUE GOODS, OR FOR ANY DIRECT, INDIRECT, INCIDENTIAL, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, EVEN IF THE SPECIFICATION LEAD OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES. IN ADDITION THE SPECIFICATION LEAD AND ITS SUPPLIERS WILL NOT BE LIABLE FOR ANY DAMAGES CLAIMED BY YOU BASED ON ANY THIRD PARTY CLAIM.

4.2 Some jurisdictions do not allow the exclusion of implied warranties, or the limitation for consequential damages, so Section 4.1 may not apply to You in whole, but in such case Section 4.1 will apply to You to the maximum extent permitted by applicable law.

5. EXPORT CONTROL

5.1 You shall follow all export control laws and regulations relating to Specification.

6. RESTRICTED RIGHTS LEGEND

6.1 Note to U.S. Government Users. The Specification is a "Commercial Items", as that term is defined at 48 C.F.R. 2. 101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation", as such terms are used in 48 C.F.R. 12.212 or 48 C.F.R. 227.7202, as applicable. Consistent with 48 C.F.R. 12.212 or 48 C.F.R. 227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software Documentation are being licensed to U.S. Government end users a) only as Commercial Items and b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.

Table of Contents

Location API

for Java™ 2 Micro Edition

Overview

Description

This document defines the JSR-179 Location API for the Java 2 Platform, Micro Edition (J2ME™) version 1.1.

Preface

This specification defines the JSR-179 Location API for J2ME, version 1.1.

Overview

This specification defines a J2ME Optional Package that enables mobile location-based applications for resource limited devices (referred to as 'terminals' in the following). The API is designed to be a compact and generic API that produces information about the present geographic location of the terminal to Java applications. The API covers obtaining information about the present geographic location and orientation of the terminal and accessing a database of known landmarks stored in the terminal.

The Location API for J2ME is designed as an Optional Package that can be used with many J2ME Profiles. The minimum platform required by this API is the J2ME Connected, Limited Device Configuration (CLDC) v1.1. The API can also be used with the J2ME Connected Device Configuration (CDC). (Note that due to using the floating point datatypes in this API, it can't be used with CLDC v1.0.)

Notation used

This specification uses the following terms as they are defined in the table below:

Term	Definition
must	The associated definition is an absolute requirement of this specification.
must not	The definition is an absolute prohibition of this specification.
should	Indicates a recommended practice. There may exist valid reasons in particular circumstances to ignore this recommendation, but the full implications must be understood and carefully weighed before choosing a different course.
should not	Indicates a non-recommended practice. There may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
may	Indicates that an item is truly optional.

Mathematical notation for defining ranges of floating point numbers is used as defined below:

`[a,b]`	a closed range from value a to value b with the end-points a and b included in the range
`(a,b)`	an open range from value a to value b with the end-points a and b not included in the range
`[a,b)`	a half-open range from value a to value b with the end-point a included and end-point b not included in the range
`(a,b]`	a half-open range from value a to value b with the end-point a not included and end-point b included in the range

Where BNF notation is used to define syntax elements, the BNF notation is used as defined in section 2.1 of RFC2616.

Report and Contact

Comments on this specification are welcome and appreciated. Comments can be sent to:
`<jsr-179-comments@jcp.org>`

Glossary

This section defines some terms used in the specification.

Assisted	Implies that the other party provides some information assisting determining the location to the party that does the final calculation. For example, for a terminal based method 'assisted' means that the network provides some assistance information. See 'terminal based' and 'network based'.
Category	Category is used to group Landmarks that are of similar type to the end user, e.g. restaurants, museums, etc. Category has an identifying unique name that identifies the category to the end user.
Coordinates	The geographical coordinates (latitude, longitude, altitude) that identify an exact location point. In this API, these are encapsulated in instances of the `Coordinates` class.
Course	Course made good, i.e. direction of the velocity vector, relative to true north.
Landmark	A known geographical location that has a name and a category associated with it. The name is a textual name that identifies that location for the end user and the category provides a grouping of similar landmarks based on some property that is meaningful to the end user. In this API, often refers to the class `Landmark`. The landmarks are usually stored in a local database in the terminal that is represented by the `LandmarkStore` class in this API.
Location	Geographical location. In this API, often refers to the class `Location` that encodes the geographical coordinates, their accuracy, the current course and speed of the terminal (if available), textual address information for the location (if available) and the timestamp when the location measurement was made.
Network based	A location method is network based if the final calculation that gives the location result is performed in the network.
Qualified Coordinates	The geographical coordinates (latitude, longitude, altitude) that identify a location combined with their accuracy information. These identify a geographical location with some uncertainty of the measurement represented by the accuracy values. In this API, these are encapsulated in instances of the `QualifiedCoordinates` class.
Terminal based	A location method is terminal based if the final calculation that gives the location result is performed in the terminal.

References

[1] *NIMA TR8350.2 World Geodetic System 1984*. It's definition and Relationships with Local Geodetic Systems. U.S. Department of Defence. January, 2000. http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf.

[2] *NMEA 0183 Interface standard, version 3.01*. The National Marine Electronics Association. January, 2002. http://www.nmea.org/pub/0183/index.html.

[3] *TS 101 Mobile Location Protocol Specification, version 3.0*. Open Mobile Alliance, Location Working Group. June, 2002. http://www.openmobilealliance.com/tech/affiliates/lif/lifindex.html.

[4] *JSR-139 Connected, Limited Device Configuration Specification, Version 1.1*. Java Community Process. March, 2003. http://www.jcp.org/en/jsr/detail?id=139.

[5] *ISO 3166-1 Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes*. International Organization for Standardization. 1997. http://www.iso.org.

[6] *JSR-118 Mobile Information Device Profile, version 2.1*. Java Community Process. June, 2006. http://www.jcp.org/en/jsr/detail?id=118.

[7] *JSR-218 Connected Device Configuration Specification, Version 1.1*. Java Community Process. August, 2005. http://www.jcp.org/en/jsr/detail?id=218.

API Reference

Package javax.microedition.location

The javax.

Interface summary	
LocationListener	The LocationListener represents a listener that receives events associated with a particular LocationProvider .
ProximityListener	This interface represents a listener to events associated with detecting proximity to some registered coordinates.

Class summary	
AddressInfo	The AddressInfo class holds textual address information about a location.
Coordinates	The Coordinates class represents coordinates as latitude-longitude-altitude values.
Criteria	The criteria used for the selection of the location provider is defined by the values in this class.
Landmark	The Landmark class represents a landmark, i.
LandmarkStore	The LandmarkStore class provides methods to store, delete and retrieve landmarks from a persistent landmark store.
Location	The Location class represents the standard set of basic location information.
LocationProvider	This is the starting point for applications using this API and represents a source of the location information.
Orientation	The Orientation class represents the physical orientation of the terminal.
QualifiedCoordinates	The QualifiedCoordinates class represents coordinates as latitude-longitude-altitude values that are associated with an accuracy value.

Exception summary	
LandmarkException	The LandmarkException is thrown when an error related to handling landmarks has occurred.
LocationException	The LocationException is thrown when a Location API specific error has occurred.

Package Documentation

The `javax.microedition.location` package contains the basic classes needed to request and get a location result.

The `LocationProvider` class represents a module that is able to determine the location of the terminal. This may be implemented using any possible location method, for example, satellite based methods like GPS, cellular network based methods, short-range positioning methods like Bluetooth Local Positioning, etc. The implementation may also combine methods in various ways to get the optimal result.

The application can specify criteria for selecting the location provider and obtain a `LocationProvider` instance that is able to fulfil these criteria as closely as possible. By using the `LocationProvider` , the application can get `Location` objects representing the location of the terminal at the time of the measurement. The application can either request a single `Location` object or be periodically updated with new `Location` objects via a `LocationListener` implemented by the application.

The location is represented by the `Location` object that contains a `QualifiedCoordinates` object representing the geographical coordinates (latitude, longitude and altitude) and information about their accuracy, a timestamp and possibly information about the speed and course of the terminal. For some location methods, the `Location` object may also contain an `AddressInfo` object that includes textual address information, e.g. a street address.

The `Location` gives the accuracy of the coordinates as the radius of a circular area indicating the 1-sigma confidence level. The 1-sigma confidence refers to the standard deviation of the distribution. Assuming a normal distribution (which is not necessarily the case), this implies that the actual location is within the circle defined by the returned point and radius at a probability of approximately 68%. The actual location may thus also be outside of the circle, which has to be taken into account when using the location information in applications.

This package also includes a database of landmarks. A landmark is a known physical location that is associated with a name representing that location for the end user. The user can store commonly used locations in the database. Examples of landmarks could be, for example, the user's home, office, etc. The default landmark database **must** be shared between all Java applications and **may** be shared with other applications in the terminal, including native applications, so that the end user has a consistent user experience across all location based applications on the terminal.

Mandatory and optional features

This API contains some options whose availability depends on the used location methods. These features are not optional in order to allow for differences in the implementations and to cause unnecessary fragmentation, but they are unavoidable due to the differences in the underlying location methods. Implementations **should** support all features that are possible with the locations methods that are used.

Mandatory features for all location methods are:

- Provide latitude and longitude coordinates and their accuracy
- Provide timestamp of the location measurement

Mandatory other API features for the terminal are:

- Support one (default) `LandmarkStore` for storing landmarks

Features whose availability depend on the used location method:

- Provide altitude information
- Provide accuracy of altitude
- Provide course and speed information
- Provide textual address information related to the location
- Provide landmark proximity events

Optional features whose availability depend on the landmark store implementation of the terminal and its possible relation to landmark stores shared with native applications:

- Creating and deleting landmark stores
- Adding and removing landmark categories

Additionally, depending on the hardware capabilities of the terminal, the following rules apply:

- If the terminal has compass hardware, the compass azimuth of the terminal orientation **must** be provided
- If the terminal has 3D orientation sensor in addition to compass hardware, the pitch and roll 3D terminal orientation information **may** be provided

In general, every implementation **must** contain all the classes, interfaces and methods as defined in this specification. Those features that are optional to implement have a defined behavior in the case the feature is not supported by the implementation.

Security of this API

Some methods in this API are defined to throw a `SecurityException` if the caller does not have the permissions needed to perform the action. This **must** be enforced by an appropriate security framework in the platform.

Using the MIDP 2.x Security Framework

If this API is implemented on the MIDP 2.x platform, the security framework of MIDP 2.x **must** be used as defined below.

The table below defines the names of the permissions used and the methods that are protected by each permission. The definition of the policy for these permissions is out of scope for this specification.

Permission name	Methods protected by this permission
javax.microedition.location.Location	LocationProvider.getLocation(), LocationProvider.setLocationListener(), LocationProvider.getLastKnownLocation()
javax.microedition.location.Orientation	Orientation.getOrientation()
javax.microedition.location.ProximityListener	LocationProvider.addProximityListener()
javax.microedition.location.LandmarkStore.read	LandmarkStore.getInstance(), LandmarkStore.listLandmarkStores()
javax.microedition.location.LandmarkStore.write	LandmarkStore.addLandmark(), LandmarkStore.deleteLandmark(), LandmarkStore.removeLandmarkFromCategory(), LandmarkStore.updateLandmark()
javax.microedition.location.LandmarkStore.category	LandmarkStore.addCategory(), LandmarkStore.deleteCategory()
javax.microedition.location.LandmarkStore.management	LandmarkStore.createLandmarkStore(), LandmarkStore.deleteLandmarkStore()

CDC and MIDP 3 security model

Since this specification can be implemented on top of CDC configuration or in devices with MIDP 3 profile, this chapter defines the security permissions for those environments.

The implementations of this specification on configurations and profiles that use the fine grained security permissions based on `java.security.Permission` security checks **must** include the class `javax.microedition.location.LocationPermission`. The table below lists the methods that **must** perform permission checks. The listing of `LocationPermission` can be found from Appendix A.

API call	Name parameter in `LocationPermission`	Action parameters in `LocationPermission`
LocationProvider.getLocation(), LocationProvider.setLocationListener(), LocationProvider.getLastKnownLocation()	"location"	"location"
Orientation.getOrientation()	"orientation"	"orientation"
LocationProvider.addProximityListener()	"proximity_listener"	"proximity"
LandmarkStore.getInstance(), LandmarkStore.listLandmarkStores()	"landmark_store"	"read"
LandmarkStore.addLandmark(), LandmarkStore.deleteLandmark(), LandmarkStore.removeLandmarkFromCategory(), LandmarkStore.updateLandmark()	"landmark_store	"write"
LandmarkStore.addCategory(), LandmarkStore.deleteCategory()	"landmark_store"	"category"
LandmarkStore.createLandmarkStore(), LandmarkStore.deleteLandmarkStore()	"landmark_store"	"management"

Identification of the Location API

To enable applications to test for the presence of the Location API and its version during runtime, a system property is defined. Platforms where this API is implemented according to this specification shall include a system property with a key `"microedition.location.version"`. When `System.getProperty` is called with this key, implementations conforming to this specification shall return the version string `"1.1"`.

Example of Usage

The following piece of code illustrates how to obtain the current location of the terminal. This piece of code illustrates obtaining the location syncronously. An application would normally perform this within a separate thread, because the `getLocation` method may block for a long time.

```
try {
       // Create a Criteria object for defining desired selection criteria
       Criteria cr = new Criteria();
       // Specify horizontal accuracy of 500 meters, leave other parameters
       // at default values.
       cr.setHorizontalAccuracy(500);
       LocationProvider lp = LocationProvider.getInstance(cr);
       // get the location, one minute timeout
       Location l = lp.getLocation(60);
       Coordinates c = l.getQualifiedCoordinates();
       if (c != null) {
           // use coordinate information
           ...
       }
   } catch (LocationException e) {
       // not able to retrive location information
       ...
   }
```

The following example illustrates how to use the `LocationListener` for subscribing to periodic location updates. This example creates a handler thread to handle the updates so that the methods on the listener would not block the platform implementation threads for a long time.

```
class MovementTracker implements LocationListener {

       float movementDistance;
       LocationProvider provider;
       Location lastValidLocation;
       UpdateHandler handler;
       boolean done;

       public MovementTracker(float movementDistance) throws LocationException {
           this.movementDistance = movementDistance;
           done = false;
           handler = new UpdateHandler();
           new Thread(handler).start();
           provider = LocationProvider.getInstance(null);
           provider.setLocationListener(this, -1, -1, -1);
       }

       public void locationUpdated(LocationProvider provider, Location location) {
           handler.handleUpdate(location);
       }

       public void providerStateChanged(LocationProvider provider, int newState) {
       }

       class UpdateHandler implements Runnable {

           private Location updatedLocation = null;
           // The run method performs the actual processing of the location
           // updates

           public void run() {
               Location locationToBeHandled = null;
               while (!done) {
                   synchronized (this) {
                       if (updatedLocation == null) {
                           try {
                               wait();
                           } catch (Exception e) {
                               // Handle interruption
                           }
                       }
                       locationToBeHandled = updatedLocation;
```

```
                    updatedLocation = null;
                }

                // The benefit of the MessageListener is here.
                // This thread could via similar triggers be
                // handling other kind of events as well in
                // addition to just receiving the location updates.
                if (locationToBeHandled != null) {
                    processUpdate(locationToBeHandled);
                }
            }
        }

        public synchronized void handleUpdate(Location update) {
            updatedLocation = update;
            notify();
        }

        private void processUpdate(Location update) {
            if (update.getQualifiedCoordinates().distance(
                 lastValidLocation.getQualifiedCoordinates()) > movementDistance) {
                // Alert user to movement...

                // Cache new position as we have moved a sufficient distance
                // from last one
                lastValidLocation = location;
            }
        }
    }
}
```

AddressInfo

AddressInfo — class in package javax.microedition.location

public class AddressInfo extends java.lang.Object

Field summary

static int	BUILDING_FLOOR
static int	BUILDING_NAME
static int	BUILDING_ROOM
static int	BUILDING_ZONE
static int	CITY
static int	COUNTRY
static int	COUNTRY_CODE
static int	COUNTY
static int	CROSSING1
static int	CROSSING2
static int	DISTRICT
static int	EXTENSION
static int	PHONE_NUMBER
static int	POSTAL_CODE
static int	STATE
static int	STREET
static int	URL

Constructor summary

AddressInfo()

Method summary

java.lang.String	getField(int field)
void	setField(int field,java.lang.String value)

Inheritance

java.lang.Object → AddressInfo

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Description

The `AddressInfo` class holds textual address information about a location. Typically the information is e.g. street address. The information is divided into fields (e.g. street, postal code, city, etc.). Defined field constants can be used to retrieve field data.

If the value of a field is not available, it is set to `null` .

The names of the fields use terms and definitions that are commonly used e.g. in the United States. Addresses for other countries should map these to the closest corresponding entities used in that country.

This class is only a container for the information. The getField(int) method returns the value set for the defined field using the setField(int, String) method. When the platform implementation returns `AddressInfo` objects, it **must** ensure that it only returns objects where the parameters have values set as described for their semantics in this class.

Below are some typical examples of addresses in different countries and how they map to the `AddressInfo` fields.

AddressInfo Field	American Example	British Example
EXTENSION	Flat 5	The Oaks
STREET	10 Washington Street	20 Greenford Court
POSTAL_CODE	12345	AB1 9YZ
CITY	Palo Alto	Cambridge
COUNTY	Santa Clara County	Cambridgeshire
STATE	California	England
COUNTRY	United States of America	United Kingdom
COUNTRY_CODE	US	GB
DISTRICT		
BUILDING_NAME		
BUILDING_FLOOR		
BUILDING_ROOM		
BUILDING_ZONE		
CROSSING1		
CROSSING2		
URL	http://www.americanurl.com	http://britishurl.co.uk
PHONE_NUMBER		

Fields

BUILDING_FLOOR

```
public static final int BUILDING_FLOOR
```

Address field denoting a building floor.

Value: 11

BUILDING_NAME

```
public static final int BUILDING_NAME
```

Address field denoting a building name.

Value: 10

BUILDING_ROOM

```
public static final int BUILDING_ROOM
```

Address field denoting a building room.

Value: 12

BUILDING_ZONE

```
public static final int BUILDING_ZONE
```

Address field denoting a building zone.

Value: 13

CITY

```
public static final int CITY
```

Address field denoting town or city name.

Value: 4

COUNTRY

```
public static final int COUNTRY
```

Address field denoting country.

Value: 7

COUNTRY_CODE

```
public static final int COUNTRY_CODE
```

Address field denoting country as a two-letter ISO 3166-1 code.

Value: 8

COUNTY

```
public static final int COUNTY
```

Address field denoting a county, which is an entity between a state and a city.

Value: 5

CROSSING1

```
public static final int CROSSING1
```

Address field denoting a street in a crossing.

Value: 14

CROSSING2

```
public static final int CROSSING2
```

Address field denoting a street in a crossing.

Value: 15

DISTRICT

```
public static final int DISTRICT
```

Address field denoting a municipal district.

Value: 9

EXTENSION

```
public static final int EXTENSION
```

Address field denoting address extension, e.g. flat number.

Value: 1

PHONE_NUMBER

```
public static final int PHONE_NUMBER
```

Address field denoting a phone number for this place.

Value: 17

POSTAL_CODE

```
public static final int POSTAL_CODE
```

Address field denoting zip or postal code.

Value: 3

STATE

```
public static final int STATE
```

Address field denoting state or province.

Value: 6

STREET

```
public static final int STREET
```

Address field denoting street name and number.

Value: 2

URL

```
public static final int URL
```

Address field denoting a URL for this place.

Value: 16

Constructors

AddressInfo()

```
public AddressInfo ()
```

Constructs an `AddressInfo` object with all the values of the fields set to `null` .

Methods

getField(int)

```
public java.lang.String getField(int field)
```

Returns the value of an address field. If the field is not available `null` is returned.

Example: `getField(AddressInfo.STREET)` might return "113 Broadway" if the location is on Broadway, New York, or `null` if not available.

Parameters

`field` - the ID of the field to be retrieved

Returns

The address field string. If the field is not set, returns `null` .

Throws

`IllegalArgumentException` - if the parameter `field` ID is not one of the constant values defined in this class

setField(int, java.lang.String)

```
public void setField(int field, java.lang.String value)
```

Sets the value of an address field.

Parameters

`field` - the ID of the field to be set
`value` - the new value for the field, `null` is used to indicate that the field has no content.

Throws

`IllegalArgumentException` - if the parameter `field` ID is not one of the constant values defined in this class

Coordinates

Coordinates — class in package javax.microedition.location

public class Coordinates extends java.lang.Object

Field summary

static int	DD_MM
static int	DD_MM_SS

Constructor summary

Coordinates(double latitude,double longitude,float altitude)

Method summary

float	azimuthTo(Coordinates to)
static java.lang.String	convert(double coordinate,int outputType)
static double	convert(java.lang.String coordinate)
float	distance(Coordinates to)
float	getAltitude()
double	getLatitude()
double	getLongitude()
void	setAltitude(float altitude)
void	setLatitude(double latitude)
void	setLongitude(double longitude)

Inheritance

java.lang.Object → Coordinates

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Direct known subclasses

QualifiedCoordinates

Description

The Coordinates class represents coordinates as latitude-longitude-altitude values. The latitude and longitude values are expressed in degrees using floating point values. The degrees are in decimal values (rather than minutes/seconds). The coordinates are given using the WGS84 datum.

This class also provides convenience methods for converting between a String coordinate representation and the double representation used in this class.

Fields

DD_MM

```
public static final int DD_MM
```

Identifier for `String` coordinate representation Degrees, Minutes, decimal fractions of a minute.

Value: 2

DD_MM_SS

```
public static final int DD_MM_SS
```

Identifier for `String` coordinate representation Degrees, Minutes, Seconds and decimal fractions of a second.

Value: 1

Constructors

Coordinates(double, double, float)

```
public Coordinates (double latitude, double longitude, float altitude)
```

Constructs a new `Coordinates` object with the values specified. The latitude and longitude parameters are expressed in degrees using floating point values. The degrees are in decimal values (rather than minutes/seconds).

The coordinate values always apply to the WGS84 datum.

The `Float.NaN` value can be used for altitude to indicate that altitude is not known.

Parameters

latitude - The latitude of the location. Valid range: [-90.0, 90.0]. Positive values indicate northern latitude and negative values southern latitude.
longitude - The longitude of the location. Valid range: [-180.0, 180.0). Positive values indicate eastern longitude and negative values western longitude.
altitude - The altitude of the location in meters, defined as height above the WGS84 ellipsoid. `Float.NaN` can be used to indicate that altitude is not known.

Throws

`IllegalArgumentException` - if an input parameter is out of the valid range

Methods

azimuthTo(Coordinates)

```
public float azimuthTo( Coordinates  to)
```

Calculates the azimuth between the two points according to the ellipsoid model of WGS84. The azimuth is relative to true north. The `Coordinates` object on which this method is called is considered the origin for the calculation and the `Coordinates` object passed as a parameter is the destination which the azimuth is calculated to. When the origin is the North pole and the destination is not the North pole, this method returns 180.0. When the origin is the South pole and the destination is not the South pole, this method returns 0.0. If the origin is equal to the destination, this method returns 0.0. The implementation **shall** calculate the result as exactly as it can. However, it is required that the result is within 1 degree of the correct result.

Parameters

to - the `Coordinates` of the destination

Returns

The azimuth to the destination in degrees. Result is within the range [0.0 ,360.0).

Throws

`NullPointerException` - if the parameter `to` is `null`

convert(double, int)

```
public static java.lang.String convert(double coordinate, int outputType)
```

Converts a `double` representation of a coordinate with decimal degrees into a `String` representation.

There are `String` syntaxes supported are the same as for the convert(String) method. The implementation **shall** provide as many significant digits for the decimal fractions as are allowed by the `String` syntax definition.

Parameters

`coordinate` - a `double` reprentation of a coordinate
`outputType` - Identifier of the type of the `String` representation wanted for output. The constant `DD_MM_SS` identifies the syntax 1 and the constant `DD_MM` identifies the syntax 2.

Returns

a `String` representation of the coordinate in a representation indicated by the parameter

Throws

`IllegalArgumentException` - if the `outputType` is not one of the two costant values defined in this class or if the `coordinate` value is not within the range [-180.0, 180.0) or is `Double.NaN`

convert(java.lang.String)

```
public static double convert(java.lang.String coordinate)
```

Converts a `String` representation of a coordinate into the `double` representation as used in this API.

There are two string syntaxes supported:

1. Degrees, minutes, seconds and decimal fractions of seconds. This is expressed as a `String` complying with the following EBNF definition where the degrees are within the range [-179, 179] and the minutes and seconds are within the range [0, 59], or the degrees is -180 and the minutes, seconds and decimal fractions are 0:

```
coordinate   = degrees ":" minutes ":" seconds "." decimalfrac |
                  degrees ":" minutes ":" seconds |
                  degrees ":" minutes
    degrees      = degreedigits | "-" degreedigits
    degreedigits = digit | nonzerodigit digit | "1" digit digit
    minutes      = minsecfirstdigit digit
    seconds      = minsecfirstdigit digit
    decimalfrac  = 1*3digit
    digit        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |"9"
    nonzerodigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
    minsecfirstdigit = "0" | "1" | "2" | "3" | "4" | "5"
```

2. Degrees, minutes and decimal fractions of minutes. This is expressed as a `String` complying with the following EBNF definition where the degrees are within the range [-179, 179] and the minutes are within the range [0, 59], or the degrees is -180 and the minutes and decimal fractions are 0:

```
coordinate   = degrees ":" minutes "." decimalfrac |
                  degrees ":" minutes
    degrees      = degreedigits | "-" degreedigits
    degreedigits = digit | nonzerodigit digit | "1" digit digit
    minutes      = minsecfirstdigit digit
    decimalfrac  = 1*5digit
```

```
digit        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |"9"
nonzerodigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
minsecfirstdigit = "0" | "1" | "2" | "3" | "4" | "5"
```

For example, for the `double` value of the coordinate `61.51d` , the corresponding syntax 1 `String` is `"61:30:36"` and the corresponding syntax 2 `String` is `"61:30.6"` .

Parameters

coordinate - a `String` in either of the two representation specified above

Returns

a `double` value with decimal degrees that matches the `String` representation given as the parameter

Throws

`IllegalArgumentException` - if the `coordinate` input parameter does not comply with the defined syntax for the specified types
`NullPointerException` - if `coordinate` is `null`

distance(Coordinates)

```
public float distance( Coordinates  to)
```

Calculates the geodetic distance between the two points according to the ellipsoid model of WGS84. Altitude is neglected from calculations.

The implementation **shall** calculate this as exactly as it can. However, it is required that the result is within 0.36% of the correct result.

Parameters

to - the `Coordinates` of the destination

Returns

the distance to the destination in meters

Throws

`NullPointerException` - if the parameter `to` is `null`

getAltitude()

```
public float getAltitude()
```

Returns the altitude component of this coordinate. Altitude is defined to mean height above the WGS84 reference ellipsoid. 0.0 means a location at the ellipsoid surface, negative values mean the location is below the ellipsoid surface, `Float.NaN` that the altitude is not available.

Returns

the altitude in meters above the reference ellipsoid

getLatitude()

```
public double getLatitude()
```

Returns the latitude component of this coordinate. Positive values indicate northern latitude and negative values southern latitude.

The latitude is given in WGS84 datum.

Returns

the latitude in degrees

getLongitude()

```
public double getLongitude()
```

Returns the longitude component of this coordinate. Positive values indicate eastern longitude and negative values western longitude.

The longitude is given in WGS84 datum.

Returns

the longitude in degrees

setAltitude(float)

```
public void setAltitude(float altitude)
```

Sets the geodetic altitude for this point.

Parameters

altitude - The altitude of the location in meters, defined as height above the WGS84 ellipsoid. 0.0 means a location at the ellipsoid surface, negative values mean the location is below the ellipsoid surface, `Float.NaN` that the altitude is not available

setLatitude(double)

```
public void setLatitude(double latitude)
```

Sets the geodetic latitude for this point. Latitude is given as a `double` expressing the latitude in degrees in the WGS84 datum.

Parameters

latitude - the latitude component of this location in degrees, valid range: [-90.0, 90.0]

Throws

`IllegalArgumentException` - if latitude is out of the valid range

setLongitude(double)

```
public void setLongitude(double longitude)
```

Sets the geodetic longitude for this point. Longitude is given as a `double` expressing the longitude in degrees in the WGS84 datum.

Parameters

longitude - the longitude of the location in degrees, valid range: [-180.0, 180.0)

Throws

`IllegalArgumentException` - if longitude is out of the valid range

Criteria

Criteria — class in package javax.microedition.location

public class Criteria extends java.lang.Object

Field summary

static int	NO_REQUIREMENT
static int	POWER_USAGE_HIGH
static int	POWER_USAGE_LOW
static int	POWER_USAGE_MEDIUM

Constructor summary

Criteria()

Method summary

int	getHorizontalAccuracy()
int	getPreferredPowerConsumption()
int	getPreferredResponseTime()
int	getVerticalAccuracy()
boolean	isAddressInfoRequired()
boolean	isAllowedToCost()
boolean	isAltitudeRequired()
boolean	isSpeedAndCourseRequired()
void	setAddressInfoRequired(boolean addressInfoRequired)
void	setAltitudeRequired(boolean altitudeRequired)
void	setCostAllowed(boolean costAllowed)
void	setHorizontalAccuracy(int accuracy)
void	setPreferredPowerConsumption(int level)
void	setPreferredResponseTime(int time)
void	setSpeedAndCourseRequired(boolean speedAndCourseRequired)
void	setVerticalAccuracy(int accuracy)

Inheritance

java.lang.Object → Criteria

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

The criteria used for the selection of the location provider is defined by the values in this class. It is up to the implementation to provide a LocationProvider that can obtain locations constrained by these values.

Instances of `Criteria` are used by the application to indicate criteria for choosing the location provider in the `LocationProvider.getInstance` method call. The implementation considers the different criteria fields to choose the location provider that best fits the defined criteria. The different criteria fields do not have any defined priority order but the implementation uses some implementation specific logic to choose the location provider that can typically best meet the defined criteria.

However, the cost criteria field is treated differently from others. If the application has set the cost field to indicate that the returned location provider is not allowed to incur financial cost to the end user, the implementation **must** guarantee that the returned location provider does not incur cost.

If there is no available location provider that is able to meet all the specified criteria, the implementation is allowed to make its own best effort selection of a location provider that is closest to the defined criteria (provided that the cost criteria is met). However, an implementation is not required to return a location provider if it does not have any available provider that is able to meet these criteria or be sufficiently close to meeting them, where the judgement of sufficiently close is an implementation dependent best effort choice. It is left up to the implementation to consider what is close enough to the specified requirements that it is worth providing the location provider to the application.

The default values for the criteria fields are specified below in the table. The default values are always the least restrictive option that will match all location providers. Default values:

Criteria field	Default value
Horizontal accuracy	NO_REQUIREMENT
Vertical accuracy	NO_REQUIREMENT
Preferred response time	NO_REQUIREMENT
Power consumption	NO_REQUIREMENT
Cost allowed	true (allowed to cost)
Speed and course required	false (not required)
Altitude required	false (not required)
Address info required	false (not required)

The implementation of this class only retains the values that are passed in using the `set*` methods. It does not try to validate the values of the parameters in any way. Applications may set any values it likes, even negative values, but the consequence may be that no matching `LocationProvider` can be created.

Fields

NO_REQUIREMENT

```
public static final int NO_REQUIREMENT
```

Constant indicating no requirements for the parameter.

Value: 0

POWER_USAGE_HIGH

```
public static final int POWER_USAGE_HIGH
```

Level indicating high power consumption allowed.

Value: 3

POWER_USAGE_LOW

```
public static final int POWER_USAGE_LOW
```

Level indicating only low power consumption allowed.

Value: 1

POWER_USAGE_MEDIUM

```
public static final int POWER_USAGE_MEDIUM
```

Level indicating average power consumption allowed.

Value: 2

Constructors

Criteria()

```
public Criteria ()
```

Constructs a `Criteria` object. All the fields are set to the default values that are specified below in the specification of the `set*` methods for the parameters.

Methods

getHorizontalAccuracy()

```
public int getHorizontalAccuracy()
```

Returns the horizontal accuracy value set in this `Criteria`.

Returns

the horizontal accuracy in meters

getPreferredPowerConsumption()

```
public int getPreferredPowerConsumption()
```

Returns the preferred power consumption.

Returns

the power consumption level, should be one of `NO_REQUIREMENT, POWER_USAGE_LOW, POWER_USAGE_MEDIUM, POWER_USAGE_HIGH`.

getPreferredResponseTime()

```
public int getPreferredResponseTime()
```

Returns the preferred maximum response time.

Returns

the maximum response time in milliseconds

getVerticalAccuracy()

```
public int getVerticalAccuracy()
```

Returns the vertical accuracy value set in this `Criteria`.

Returns

the accuracy in meters

isAddressInfoRequired()

```
public boolean isAddressInfoRequired()
```

Returns whether the location provider should be able to determine textual address information.

Returns

whether the location provider should be able to normally provide textual address information, `true` means that it should be able, `false` means that this is not required.

isAllowedToCost()

```
public boolean isAllowedToCost()
```

Returns the preferred cost setting.

Returns

the preferred cost setting, `true` if it is allowed to cost, `false` if it must be free of charge

isAltitudeRequired()

```
public boolean isAltitudeRequired()
```

Returns whether the location provider should be able to determine altitude.

Returns

whether the location provider should be able to determine altitude, `true` means that it should be able, `false` means that this is not required.

isSpeedAndCourseRequired()

```
public boolean isSpeedAndCourseRequired()
```

Returns whether the location provider should be able to determine speed and course.

Returns

whether the location provider should be able to determine speed and course. `true` means that it should be able, `false` means that this is not required.

setAddressInfoRequired(boolean)

```
public void setAddressInfoRequired(boolean addressInfoRequired)
```

Sets whether the location provider should be able to determine textual address information. Setting this criteria to true implies that a location provider should be selected that is capable of providing the textual address information. This does not mean that every returned location instance necessarily will have all the address information filled in, though.

Default is `false` .

Parameters

`addressInfoRequired` - If set to `true` , the `LocationProvider` is required to be able to normally determine the textual address information. If set the `false` , the textual address information is not required.

setAltitudeRequired(boolean)

```
public void setAltitudeRequired(boolean altitudeRequired)
```

Sets whether the location provider should be able to determine altitude. Default is `false` .

Parameters

altitudeRequired - If set to `true` , the `LocationProvider` is required to be able to normally determine the altitude. If set the `false` , the altitude is not required.

setCostAllowed(boolean)

```
public void setCostAllowed(boolean costAllowed)
```

Sets the preferred cost setting.

Sets whether the requests for location determination is allowed to incur any financial cost to the user of the terminal.

The default is `true` , i.e. the method is allowed to cost.

Note that the platform implementation may not always be able to know if a location method implies cost to the end user or not. If the implementation doesn't know, it **must** assume that it may cost. When this criteria is set to `false` , the implementation **may** only return a `LocationProvider` of which it is certain that using it for determining the location does not cause a per usage cost to the end user.

Parameters

costAllowed - `false` if location determination is not allowed to cost, `true` if it is allowed to cost

setHorizontalAccuracy(int)

```
public void setHorizontalAccuracy(int accuracy)
```

Sets the desired horizontal accuracy preference. Accuracy is measured in meters. The preference indicates maximum allowed typical 1-sigma standard deviation for the location method. Default is `NO_REQUIREMENT` , meaning no preference on horizontal accuracy.

Parameters

accuracy - the preferred horizontal accuracy in meters

setPreferredPowerConsumption(int)

```
public void setPreferredPowerConsumption(int level)
```

Sets the preferred maximum level of power consumption.

These levels are inherently indeterminable and depend on many factors. It is the judgement of the implementation that defines a positioning method as consuming low power or high power. Default is `NO_REQUIREMENT` , meaning power consumption is not a quality parameter.

Parameters

level - the preferred maximum level of power consumption, should be one of `NO_REQUIREMENT, POWER_USAGE_LOW, POWER_USAGE_MEDIUM, POWER_USAGE_HIGH` .

setPreferredResponseTime(int)

```
public void setPreferredResponseTime(int time)
```

Sets the desired maximum response time preference. This value is typically used by the implementation to determine a location method that typically is able to produce the location information within the defined time. Default is `NO_REQUIREMENT` , meaning no response time constraint.

Parameters

time - the preferred time constraint and timeout value in milliseconds

setSpeedAndCourseRequired(boolean)

```
public void setSpeedAndCourseRequired(boolean speedAndCourseRequired)
```

Sets whether the location provider should be able to determine speed and course. Default is `false` .

Parameters

speedAndCourseRequired - If set to `true` , the `LocationProvider` is required to be able to normally determine the speed and course. If set the `false` , the speed and course are not required.

setVerticalAccuracy(int)

```
public void setVerticalAccuracy(int accuracy)
```

Sets the desired vertical accuracy preference. Accuracy is measured in meters. The preference indicates maximum allowed typical 1-sigma standard deviation for the location method. Default is `NO_REQUIREMENT` , meaning no preference on vertical accuracy.

Parameters

accuracy - the preferred vertical accuracy in meters

Landmark

Landmark — class in package `javax.microedition.location`

```
public class Landmark extends java.lang.Object
```

Constructor summary

`Landmark(java.lang.String name,java.lang.String description,QualifiedCoordinates coordinates,AddressInfo addressInfo)`

Method summary

`AddressInfo`	`getAddressInfo()`
`java.lang.String`	`getDescription()`
`java.lang.String`	`getName()`
`QualifiedCoordinates`	`getQualifiedCoordinates()`
`void`	`setAddressInfo(AddressInfo addressInfo)`
`void`	`setDescription(java.lang.String description)`
`void`	`setName(java.lang.String name)`
`void`	`setQualifiedCoordinates(QualifiedCoordinates coordinates)`

Inheritance

`java.lang.Object` → `Landmark`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

The `Landmark` class represents a landmark, i.e. a known location with a name. A landmark has a name by which it is known to the end user, a textual description, QualifiedCoordinates and optionally AddressInfo .

This class is only a container for the information. The constructor does not validate the parameters passed in but just stores the values, except the name field is never allowed to be `null` . The `get*` methods return the values passed in the constructor or if the values are later modified by calling the `set*` methods, the `get*` methods return the modified values. The `QualifiedCoordinates` object inside the landmark is a mutable object and the `Landmark` object holds only a reference to it. Therefore, it is possible to modify the `QualifiedCoordinates` object inside the `Landmark` object by calling the `set*` methods in the `QualifiedCoordinates` object. However, any such dynamic modifications affect only the `Landmark` object instance, but **must not** automatically update the persistent landmark information in the landmark store. The LandmarkStore.updateLandmark method is the only way to commit the modifications to the persistent landmark store.

When the platform implementation returns `Landmark` objects, it **must** ensure that it only returns objects where the parameters have values set as described for their semantics in this class.

In version 2.0 of the Location API, new fields are added to the landmark. Thses fields are extra info and timestamp. With the extra info fields tha API implementation is able to provide additional implementation dependant information to the application. Timestamp field is added to better track when the data in a landmark has been changed. Version 2.0 also provides a new constructor where values for the new fields are also provided.

Constructors

Landmark(java.lang.String, java.lang.String, QualifiedCoordinates , AddressInfo)

```
public Landmark (java.lang.String name, java.lang.String description,
QualifiedCoordinates  coordinates,  AddressInfo  addressInfo)
```

Constructs a new `Landmark` object with the values specified.

Parameters

name - the name of the landmark
description - description of the landmark, may be `null` if not available
coordinates - the `Coordinates` of the landmark, may be `null` if not known
addressInfo - the textual address information of the landmark, may be `null` if not known

Throws

`NullPointerException` - if the `name` is `null`

Methods

getAddressInfo()

```
public  AddressInfo  getAddressInfo()
```

Gets the `AddressInfo` of the landmark.

Returns

the `AddressInfo` of the landmark, `null` if not available

getDescription()

```
public java.lang.String getDescription()
```

Gets the landmark description.

Returns

the description of the landmark, `null` if not available

getName()

```
public java.lang.String getName()
```

Gets the landmark name.

Returns

the name of the landmark

getQualifiedCoordinates()

```
public  QualifiedCoordinates  getQualifiedCoordinates()
```

Gets the `QualifiedCoordinates` of the landmark.

Returns

the `QualifiedCoordinates` of the landmark, `null` if not available

setAddressInfo(AddressInfo)

```
public void setAddressInfo( AddressInfo  addressInfo)
```

Sets the AddressInfo of the landmark.

Parameters

addressInfo - the AddressInfo of the landmark, null may be passed in to indicate that description is not available

setDescription(java.lang.String)

```
public void setDescription(java.lang.String description)
```

Sets the description of the landmark.

Parameters

description - description for the landmark, null may be passed in to indicate that description is not available

setName(java.lang.String)

```
public void setName(java.lang.String name)
```

Sets the name of the landmark.

Parameters

name - name for the landmark

Throws

NullPointerException - if the parameter is null

setQualifiedCoordinates(QualifiedCoordinates)

```
public void setQualifiedCoordinates( QualifiedCoordinates  coordinates)
```

Sets the QualifiedCoordinates of the landmark.

Parameters

coordinates - the qualified coordinates of the landmark, null may be passed in to indicate that description is not available

LandmarkException

LandmarkException — exception class in package `javax.microedition.location`

```
public class LandmarkException extends java.lang.Exception
```

Constructor summary

`LandmarkException()`
`LandmarkException(java.lang.String s)`

Inheritance

`java.lang.Object` → `java.lang.Throwable` → `java.lang.Exception` → `LandmarkException`

Methods inherited from class `java.lang.Throwable`
`fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getMessage`, `getStackTrace`, `initCause`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Description

The `LandmarkException` is thrown when an error related to handling landmarks has occurred.

Constructors

LandmarkException()

```
public LandmarkException ()
```

Constructs a `LandmarkException` with no detail message.

LandmarkException(java.lang.String)

```
public LandmarkException (java.lang.String s)
```

Constructs a `LandmarkException` with the specified detail message.

Parameters

s - the detailed message

LandmarkStore

LandmarkStore — class in package `javax.microedition.location`

```
public class LandmarkStore extends java.lang.Object
```

Method summary

void	addCategory(java.lang.String categoryName)
void	addLandmark(Landmark landmark,java.lang.String category)
static void	createLandmarkStore(java.lang.String storeName)
void	deleteCategory(java.lang.String categoryName)
void	deleteLandmark(Landmark landmark)
static void	deleteLandmarkStore(java.lang.String storeName)
java.util.Enumeration	getCategories()
static LandmarkStore	getInstance(java.lang.String storeName)
java.util.Enumeration	getLandmarks(java.lang.String category, java.lang.String name)
java.util.Enumeration	getLandmarks()
java.util.Enumeration	getLandmarks(java.lang.String category,double minLatitude, double maxLatitude,double minLongitude,double maxLongitude)
static java.lang.String[]	listLandmarkStores()
void	removeLandmarkFromCategory(Landmark lm, java.lang.String category)
void	updateLandmark(Landmark landmark)

Inheritance

`java.lang.Object` → `LandmarkStore`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

The `LandmarkStore` class provides methods to store, delete and retrieve landmarks from a persistent landmark store. There is one default landmark store and there may be multiple other named landmark stores. The implementation may support creating and deleting landmark stores by the application. All landmark stores **must** be shared between all Java ME applications and **may** be shared with native applications in the terminal. Named landmark stores have unique names in this API. If the underlying implementation allows multiple landmark stores with the same name, it must present them with unique names in the API e.g. by adding some postfix to those names that have multiple instances in order to differentiate them.

Because native landmark stores may be stored as files in a file system and file systems have sometimes limitations for the allowed characters in file names, the implementations **must** support all other Unicode characters in landmark store names except the following list:

- 0x0000...0x001F control characters
- 0x005C '\'
- 0x002F '/'
- 0x003A ':'
- 0x002A '*'
- 0x003F '?'
- 0x0022 '"'
- 0x003C '<'
- 0x003E '>'
- 0x007C '|'
- 0x007F...0x009F control characters
- 0xFEFF Byte-order-mark
- 0xFFF0...0xFFFF

Support for the listed characters is not required and therefore applications are strongly encouraged not to use the characters listed above in landmark store names in order to ensure interoperability of the application on different platform implementations.

The Landmark objects have a name and may be placed in a category or several categories. The category is intended to group landmarks that are of similar type to the end user, e.g. restaurants, museums, etc. The landmark names are strings that identify the landmark to the end user. The category names describe the category to the end user. The language used in the names may be any and depends on the preferences of the end user. The names of the categories are unique within a `LandmarkStore` . However, the names of the landmarks are not guaranteed to be unique. `Landmark` objects with the same name can appear in multiple categories or even several `Landmark` objects with the same name in the same category.

The `Landmark` objects returned from the `getLandmarks` methods in this class shall guarantee that the application can read a consistent set of the landmark data valid at the time of obtaining the object instance, even if the landmark information in the store is modified subsequently by this or some other application.

The `Landmark` object instances can be in two states:

- initially constructed by an application
- belongs to a `LandmarkStore`

A `Landmark` object belongs to a `LandmarkStore` if it has been obtained from the `LandmarkStore` using `getLandmarks` or if it has been added to the `LandmarkStore` using `addLandmark` . A `Landmark` object is initially constructed by an application when it has been constructed using the constructor but has not been added to a `LandmarkStore` using `addLandmark` .

Note that the term "belongs to a `LandmarkStore` " is defined above in a way that "belong to a `LandmarkStore` " has an different meaning than the landmark "being currently stored in a `LandmarkStore` ". According to the above definition, a `Landmark` object instance may be in a state where it is considered to "belong to a `LandmarkStore` " even when it is not stored in that `LandmarkStore` (e.g. if the landmark is deleted from the `LandmarkStore` using `deleteLandmark` method, the `Landmark` object instance still is considered to "belong to this `LandmarkStore` ").

The landmark stores created by an application and landmarks added in landmark stores persist even if the application itself is deleted from the terminal.

Accessing the landmark store may cause a `SecurityException` , if the calling application does not have the required permissions. The permissions to read and write (including add and delete) landmarks are distinct. An application having e.g. a permission to read landmarks would not necessarily have the permission to delete them. The permissions (names etc.) for the MIDP 2.0 security framework are defined elsewhere in this specification.

Methods

addCategory(java.lang.String)

```
public void addCategory(java.lang.String categoryName) throws
javax.microedition.location.LandmarkException, java.io.IOException
```

Adds a category to this `LandmarkStore` .

All implementations must support names that have length up to and including 32 characters. If the provided name is longer it may be truncated by the implementation if necessary.

Parameters

`categoryName` - name for the category to be added

Throws

`IllegalArgumentException` - if a category with the specified name already exists
`NullPointerException` - if the `categoryName` is `null`
`LandmarkException` - if this `LandmarkStore` does not support adding new categories
`IOException` - if an I/O error occurs or there are no resources to add a new category
`SecurityException` - if the application does not have the permission to manage categories
`javax.microedition.location.LandmarkException`
`java.io.IOException`

addLandmark(Landmark , java.lang.String)

```
public void addLandmark( Landmark  landmark, java.lang.String category) throws
java.io.IOException
```

Adds a landmark to the specified group in the landmark store.

If some textual `String` field inside the landmark object is set to a value that is too long to be stored, the implementation is allowed to automatically truncate fields that are too long.

However, the name field **must not** be truncated. Every implementation shall be able to support name fields that are 32 characters or shorter. Implementations may support longer names but are not required to. If an application tries to add a `Landmark` with a longer name field than the implementation can support, `IllegalArgumentException` is thrown.

When the landmark store is empty, every implementation is required to be able to store a landmark where each `String` field is set to a 30 character long `String` .

If the `Landmark` object that is passed as a parameter is an instance that belongs to this `LandmarkStore` , the same `Landmark` instance will be added to the specified category in addition to the category/categories which it already belongs to. If the landmark already belongs to the specified category, this method returns with no effect. If the landmark has been deleted after obtaining it from `getLandmarks` , it will be added back when this method is called.

If the `Landmark` object that is passed as a parameter is an instance initially constructed by the application using the constructor or an instance that belongs to a different `LandmarkStore` , a new landmark will be created in this `LandmarkStore` and it will belong initially to only the category specified in the category parameter. After this method call, the `Landmark` object that is passed as a parameter belongs to this `LandmarkStore` .

Parameters

`landmark` - the landmark to be added
`category` - category where the landmark is added, `null` can be used to indicate that the landmark does not belong to a category

Throws

`IllegalArgumentException` - if the landmark has a longer name field than the implementation can support or if the category is not `null` or one of the categories defined in this `LandmarkStore`
`NullPointerException` - if the `landmark` parameter is `null`

IOException - if an I/O error happened when accessing the landmark store or if there are no resources available to store this landmark
SecurityException - if the application is not allowed to add landmarks
java.io.IOException

createLandmarkStore(java.lang.String)

```
public static void createLandmarkStore(java.lang.String storeName) throws
java.io.IOException, javax.microedition.location.LandmarkException
```

Creates a new landmark store with a specified name.

All landmark stores are shared between all Java ME applications and may be shared with native applications. Implementations may support creating landmark stores on a removable media. However, the Java application is not able to directly choose where the landmark store is stored, if the implementation supports several storage media. The implementation of this method may e.g. prompt the end user to make the choice if the implementation supports several storage media. If the landmark store is stored on a removable media, this media might be removed by the user possibly at any time causing it to become unavailable.

A newly created landmark store does not contain any landmarks.

Note that the landmark store name **may** be modified by the implementation when the store is created, e.g. by adding an implementation specific post-fix to differentiate stores on different storage drives as described in the class overview. Therefore, the application needs to use the `listLandmarkStores` method to discover the form the name was stored as. However, when creating stores to the default storage location, it is recommended that the implementation does not modify the store name but preserves it in the form it was passed to this method. It is strongly recommended that this method is implemented as character case preserving for the store name.

Parameters

storeName - the name of the landmark store to create

Throws

NullPointerException - if the storeName is null
IllegalArgumentException - if the name is too long or if a landmark store with the specified name already exists
IOException - if the landmark store could not be created due to an I/O error
SecurityException - if the application does not have permissions to create a new landmark store
LandmarkException - if the implementation does not support creating new landmark stores
java.io.IOException
javax.microedition.location.LandmarkException

deleteCategory(java.lang.String)

```
public void deleteCategory(java.lang.String categoryName) throws
javax.microedition.location.LandmarkException, java.io.IOException
```

Removes a category from this LandmarkStore . The category will be removed from all landmarks that are in that category. However, this method will not remove any of the landmarks, only the associated category information from the landmarks. If a category with the supplied name does not exist in this LandmarkStore , the method returns silently with no error.

Parameters

categoryName - name for the category to be removed

Throws

NullPointerException - if the categoryName is null
LandmarkException - if this LandmarkStore does not support deleting categories
IOException - if an I/O error occurs
SecurityException - if the application does not have the permission to manage categories
javax.microedition.location.LandmarkException
java.io.IOException

deleteLandmark(Landmark)

```
public void deleteLandmark( Landmark  landmark) throws java.io.IOException,
javax.microedition.location.LandmarkException
```

Deletes a landmark from this `LandmarkStore` . This method removes the specified landmark from all categories and deletes the information from this `LandmarkStore` .

The `Landmark` instance passed as the parameter must be an instance that belongs to this `LandmarkStore` .

If the `Landmark` belongs to this `LandmarkStore` but has already been deleted from this `LandmarkStore` , then the request is silently ignored and the method call returns with no error.

Note that `LandmarkException` is thrown if the `Landmark` instance does not belong to this `LandmarkStore` , and this is different from not being stored currently in this `LandmarkStore` .

Parameters

`landmark` - the landmark to be deleted

Throws

`SecurityException` - if the application is not allowed to delete the landmark
`LandmarkException` - if the landmark instance passed as the parameter does not belong to this `LandmarkStore`
`IOException` - if an I/O error happened when accessing the landmark store
`NullPointerException` - if the `landmark` is `null`
`java.io.IOException`
`javax.microedition.location.LandmarkException`

deleteLandmarkStore(java.lang.String)

```
public static void deleteLandmarkStore(java.lang.String storeName) throws
java.io.IOException, javax.microedition.location.LandmarkException
```

Delete a landmark store with a specified name. All the landmarks and categories defined in the named landmark store are irrevocably removed.

If a landmark store with the specified name does not exist, this method returns silently without any error.

Note that the landmark store names **may** be handled as either case-sensitive or case-insensitive (e.g. Unicode collation algorithm level 2). Therefore, the implementation **must** accept the names in the form returned by `listLandmarkStores` and **may** accept the name in other variations of character case.

Parameters

`storeName` - the name of the landmark store to delete

Throws

`NullPointerException` - if the `storeName` is `null` (the default landmark store can not be deleted)
`IOException` - if the landmark store could not be deleted due to an I/O error
`SecurityException` - if the appliction does not have permissions to delete a landmark store
`LandmarkException` - if the implementation does not support deleting landmark stores
`java.io.IOException`
`javax.microedition.location.LandmarkException`

getCategories()

```
public java.util.Enumeration getCategories()
```

Returns the category names that are defined in this `LandmarkStore` . The language and locale used for these names depends on the implementation and end user settings. The names shall be such that they can be displayed to the end user and have a meaning to the end user.

Returns

An `java.util.Enumeration` containing `String` s representing the category names. If there are no categories defined in this `LandmarkStore` , an `Enumeration` with no entries is returned.

getInstance(java.lang.String)

```
public static  LandmarkStore  getInstance(java.lang.String storeName)
```

Gets a `LandmarkStore` instance for storing, deleting and retrieving landmarks. There **must** be one default landmark store and there may be other landmark stores that can be accessed by name.

Note that the landmark store names **may** be handled as either case-sensitive or case-insensitive (e.g. Unicode collation algorithm level 2). Therefore, the implementation **must** accept the names in the form returned by `listLandmarkStores` and **may** accept the name in other variations of character case.

Parameters

`storeName` - the name of the landmark store to open, if `null` , the default landmark store will be returned

Returns

the `LandmarkStore` object representing the specified landmark store or `null` if a landmark store with the specified name does not exist

Throws

`SecurityException` - if the application does not have a permission to read landmark stores

getLandmarks(java.lang.String, java.lang.String)

```
public java.util.Enumeration getLandmarks(java.lang.String category, java.lang.String
name) throws java.io.IOException
```

Gets the landmarks from the storage where the category and/or name matches the given parameters.

Parameters

`category` - the category of the landmark, `null` implies a wildcard that matches all categories
`name` - the name of the desired landmark, `null` implies a wildcard that matches all the names within the category indicated by the category parameter

Returns

an `Enumeration` containing all the matching landmarks or `null` if no landmark matched the given parameters

Throws

`IOException` - if an I/O error happened when accessing the landmark store
`java.io.IOException`

getLandmarks()

```
public java.util.Enumeration getLandmarks() throws java.io.IOException
```

Lists all landmarks stored in the store.

Returns

an `java.util.Enumeration` object containing `Landmark` objects representing all the landmarks stored in this `LandmarkStore` or `null` if there are no landmarks in the store

Throws

`IOException` - if an I/O error happened when accessing the landmark store

`java.io.IOException`

getLandmarks(java.lang.String, double, double, double, double)

```
public java.util.Enumeration getLandmarks(java.lang.String category, double minLatitude,
double maxLatitude, double minLongitude, double maxLongitude) throws java.io.IOException
```

Lists all the landmarks that are within an area defined by bounding minimum and maximum latitude and longitude and belong to the defined category, if specified. The bounds are considered to belong to the area.

If `minLongitude <= maxLongitude` , this area covers the longitude range `[minLongitude, maxLongitude]` . If `minLongitude > maxLongitude` , this area covers the longitude range `[-180.0, maxLongitude]` and `[minLongitude, 180.0)` .

For latitude, the area covers the latitude range `[minLatitude, maxLatitude]` .

Parameters

`category` - the category of the landmark. `null` implies a wildcard that matches all categories
`minLatitude` - minimum latitude of the area, must be within the range [-90.0, 90.0]
`maxLatitude` - maximum latitude of the area, must be within the range [minLatitude, 90.0]
`minLongitude` - minimum longitude of the area, must be within the range [-180.0, 180.0)
`maxLongitude` - maximum longitude of the area, must be within the range [-180.0, 180.0)

Returns

an `Enumeration` containing all the matching landmarks or `null` if no landmark matched the given parameters

Throws

`IOException` - if an I/O error happened when accessing the landmark store
`IllegalArgumentException` - if the `minLongitude` or `maxLongitude` is out of the range [-180.0, 180.0), or `minLatitude` or `maxLatitude` is out of the range[-90.0,90.0], or if `minLatitude > maxLatitude`
`java.io.IOException`

listLandmarkStores()

```
public static java.lang.String[] listLandmarkStores() throws java.io.IOException
```

Lists the names of all the available landmark stores.

The default landmark store is obtained from `getInstance` by passing `null` as the parameter. The `null` name for the default landmark store is not included in the list returned by this method. If there are no named landmark stores, other than the default landmark store, this method returns `null` .

The store names must be returned in a form that is directly usable as input to `getInstance` and `deleteLandmarkStore` .

Returns

an array of landmark store names

Throws

`SecurityException` - if the application does not have the permission to access landmark stores
`IOException` - if an I/O error occurred when trying to access the landmark stores
`java.io.IOException`

removeLandmarkFromCategory(Landmark , java.lang.String)

```
public void removeLandmarkFromCategory( Landmark  lm, java.lang.String category) throws
java.io.IOException
```

Removes the named landmark from the specified category.

The Landmark instance passed in as the parameter must be an instance that belongs to this LandmarkStore .

If the Landmark is not found in this LandmarkStore in the specified category or if the parameter is a Landmark instance that does not belong to this LandmarkStore , then the request is silently ignored and the method call returns with no error. The request is also silently ignored if the specified category does not exist in this LandmarkStore .

The landmark is only removed from the specified category but the landmark information is retained in the store. If the landmark no longer belongs to any category, it can still be obtained from the store by passing null as the category to getLandmarks .

Parameters

lm - the landmark to be removed
category - the category from which it will be removed

Throws

SecurityException - if the application is not allowed to delete the landmark
IOException - if an I/O error happened when accessing the landmark store
NullPointerException - if either parameter is null
java.io.IOException

updateLandmark(Landmark)

```
public void updateLandmark( Landmark  landmark) throws java.io.IOException,
javax.microedition.location.LandmarkException
```

Updates the information about a landmark. This method only updates the information about a landmark and does not modify the categories the landmark belongs to.

The Landmark instance passed in as the parameter must be an instance that belongs to this LandmarkStore .

This method can not be used to add a new landmark to the store.

Parameters

landmark - the landmark to be updated

Throws

NullPointerException - if the landmark is null
IllegalArgumentException - if the landmark has a longer name field than the implementation can support
LandmarkException - if the landmark instance passed as the parameter does not belong to this LandmarkStore or does not exist in the store any more
IOException - if an I/O error happened when accessing the landmark store
SecurityException - if the application is not allowed to update the landmark
java.io.IOException
javax.microedition.location.LandmarkException

Location

Location — class in package javax.microedition.location

public class Location extends java.lang.Object

Field summary

static int	MTA_ASSISTED
static int	MTA_UNASSISTED
static int	MTE_ANGLEOFARRIVAL
static int	MTE_CELLID
static int	MTE_SATELLITE
static int	MTE_SHORTRANGE
static int	MTE_TIMEDIFFERENCE
static int	MTE_TIMEOFARRIVAL
static int	MTY_NETWORKBASED
static int	MTY_TERMINALBASED

Constructor summary

Location()

Method summary

AddressInfo	getAddressInfo()
float	getCourse()
java.lang.String	getExtraInfo(java.lang.String mimetype)
int	getLocationMethod()
QualifiedCoordinates	getQualifiedCoordinates()
float	getSpeed()
long	getTimestamp()
boolean	isValid()

Inheritance

java.lang.Object → Location

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Description

The `Location` class represents the standard set of basic location information. This includes the timestamped coordinates, accuracy, speed, course, and information about the positioning method used for the location, plus an optional textual address.

The location method is indicated using a bit field. The individual bits are defined using constants in this class. This bit field is a bitwise combination of the location method technology bits (`MTE_*`), method type (`MTY_*`) and method assistance information (`MTA_*`). All other bits in the 32 bit integer than those that have defined constants in this class are reserved and **must not** be set by implementations (i.e. these bits **must** be 0).

A `Location` object may be either 'valid' or 'invalid'. The validity can be queried using the isValid() method. A valid `Location` object represents a location with valid coordinates and the getQualifiedCoordinates() method **must** return there coordinates. An invalid `Location` object does not have valid coordinates, but the extra info that is obtained from the getExtraInfo(String) method can provide information about the reason why it was not possible to provide a valid `Location` . For an invalid `Location` object, the `getQualifiedCoordinates` method may return either `null` or some coordinates where the information is not necessarily fully correct. The periodic location updates to the LocationListener may return invalid Location objects if it isn't possible to determine the location.

This class is only a container for the information. When the platform implementation returns `Location` objects, it **must** ensure that it only returns objects where the parameters have values set as described for their semantics in this class.

Fields

MTA_ASSISTED

```
public static final int MTA_ASSISTED
```

Location method is assisted by the other party (Terminal assisted for Network based, Network assisted for terminal based).

`MTA_ASSISTED = 0x00040000`

Value: 262144

MTA_UNASSISTED

```
public static final int MTA_UNASSISTED
```

Location method is unassisted. This bit and `MTA_ASSISTED` bit **must not** both be set. Only one of these bits may be set or neither to indicate that the assistance information is not known.

`MTA_UNASSISTED = 0x00080000`

Value: 524288

MTE_ANGLEOFARRIVAL

```
public static final int MTE_ANGLEOFARRIVAL
```

Location method Angle of Arrival for cellular / terrestrial RF system.

`MTE_ANGLEOFARRIVAL = 0x00000020`

Value: 32

MTE_CELLID

```
public static final int MTE_CELLID
```

Location method Cell-ID for cellular (in GSM, this is the same as CGI, Cell Global Identity).

`MTE_CELLID = 0x00000008`

Value: 8

MTE_SATELLITE

```
public static final int MTE_SATELLITE
```

Location method using satellites (for example, Global Positioning System (GPS)).

```
MTE_SATELLITE = 0x00000001
```

Value: 1

MTE_SHORTRANGE

```
public static final int MTE_SHORTRANGE
```

Location method Short-range positioning system (for example, Bluetooth LP).

```
MTE_SHORTRANGE = 0x00000010
```

Value: 16

MTE_TIMEDIFFERENCE

```
public static final int MTE_TIMEDIFFERENCE
```

Location method Time Difference for cellular / terrestrial RF system (for example, Enhanced Observed Time Difference (E-OTD) for GSM).

```
MTE_TIMEDIFFERENCE = 0x00000002
```

Value: 2

MTE_TIMEOFARRIVAL

```
public static final int MTE_TIMEOFARRIVAL
```

Location method Time of Arrival (TOA) for cellular / terrestrial RF system.

```
MTE_TIMEOFARRIVAL = 0x00000004
```

Value: 4

MTY_NETWORKBASED

```
public static final int MTY_NETWORKBASED
```

Location method is of type network based. This means that the final location result is calculated in the network. This bit and `MTY_TERMINALBASED` bit **must not** both be set. Only one of these bits may be set or neither to indicate that it is not known where the result is calculated.

```
MTY_NETWORKBASED = 0x00020000
```

Value: 131072

MTY_TERMINALBASED

```
public static final int MTY_TERMINALBASED
```

Location method is of type terminal based. This means that the final location result is calculated in the terminal.

```
MTY_TERMINALBASED = 0x00010000
```

Value: 65536

Constructors

Location()

```
protected Location ()
```

A protected constructor for the `Location` to allow implementations of `LocationProviders` to construct the `Location` instances.

This method is not intended to be used by applications.

This constructor sets the fields to implementation specific default values. Location providers are expected to set the fields to the correct values after constructing the object instance.

Methods

getAddressInfo()

```
public  AddressInfo  getAddressInfo()
```

Returns the `AddressInfo` associated with this `Location` object. If no address is available, `null` is returned.

Returns

an `AddressInfo` associated with this `Location` object

getCourse()

```
public float getCourse()
```

Returns the terminal's course made good in degrees relative to true north. The value is always in the range [0.0,360.0) degrees.

Returns

the terminal's course made good in degrees relative to true north or `Float.NaN` if the course is not known

getExtraInfo(java.lang.String)

```
public java.lang.String getExtraInfo(java.lang.String mimetype)
```

Returns extra information about the location. This method is intended to provide location method specific extra information that applications that are aware of the used location method and information format are able to use.

A MIME type is used to identify the type of the extra information when requesting it. If the implementation supports this type, it returns the extra information as a `String` encoded according to format identified by the MIME type. If the implementation does not support this type, the method returns `null` .

This specification does not require implementations to support any extra information type.

The following MIME types are defined here together with their definitions in order to ensure interoperability of implementations wishing to use these types. The definition of these types here is not an indication that these formats are preferred over any other format not defined here.

When the MIME type is `"application/X-jsr179-location-nmea"` , the returned string **shall** be a valid sequence of NMEA sentences formatted according to the syntax specified in the NMEA 0183 v3.1 specification [NMEA]. These sentences **shall** represent the set of NMEA sentences that are related to this location at the time this location was created.

When the MIME type is `"application/X-jsr179-location-lif"` , the returned string **shall** contain an XML formatted document containing the "pd" element defined in the LIF Mobile Location Protocol TS 101 v3.0.0 [MLP] as the root element of the document.

When the MIME type is `"text/plain"` , the returned `String` **shall** contain textual extra information that can be displayed to the end user.

Parameters

mimetype - the MIME type of the requested extra information

Returns

`String` encoded according to the format identified by the MIME type defined in the parameter, `null` if the information for the requested MIME type is not available or not supported by this implementation.

getLocationMethod()

```
public int getLocationMethod()
```

Returns information about the location method used. The returned value is a bitwise combination (OR) of the method technology, method type and assistance information. The method technology values are defined as constant values named `MTE_*` in this class, the method type values are named `MTY_*` and assistance information values are named `MTA_*` .

For example, if the location method used is terminal based, network assisted E-OTD, the value 0x00050002 (= `MTY_TERMINALBASED | MTA_ASSISTED | MTE_TIMEDIFFERENCE`) would be returned.

If the location is determined by combining several location technologies, the returned value may have several `MTE_*` bits set.

If the used location method is unknown, the returned value **must** have all the bits set to zero.

Only bits that have defined constants within this class are allowed to be used. Other bits are reserved and **must** be set to 0.

Returns

a bitfield identifying the used location method

getQualifiedCoordinates()

```
public  QualifiedCoordinates  getQualifiedCoordinates()
```

Returns the coordinates of this location and their accuracy.

Returns

a `QualifiedCoordinates` object, if the coordinates are not known, returns `null`

getSpeed()

```
public float getSpeed()
```

Returns the terminal's current ground speed in meters per second (m/s) at the time of measurement. The speed is always a non-negative value. Note that unlike the coordinates, speed does not have an associated accuracy because the methods used to determine the speed typically are not able to indicate the accuracy.

Returns

the current ground speed in m/s for the terminal or `Float.NaN` if the speed is not known

getTimestamp()

```
public long getTimestamp()
```

Returns the timestamp at which the data was collected. This timestamp should represent the point in time when the measurements were made. Implementations make best effort to set the timestamp as close to this point in time as possible. The time returned is the time of the local clock in the terminal in milliseconds using the same clock and same time representation as `System.currentTimeMillis()` .

Returns

a timestamp representing the time

isValid()

```
public boolean isValid()
```

Returns whether this `Location` instance represents a valid location with coordinates or an invalid one where all the data, especially the latitude and longitude coordinates, may not be present.

A valid `Location` object contains valid coordinates whereas an invalid `Location` object may not contain valid coordinates but may contain other information via the `getExtraInfo()` method to provide information on why it was not possible to provide a valid `Location` object.

Returns

a boolean value with `true` indicating that this `Location` instance is valid and `false` indicating an invalid `Location` instance

LocationException

LocationException — exception class in package `javax.microedition.location`

```
public class LocationException extends java.lang.Exception
```

Constructor summary

`LocationException()`
`LocationException(java.lang.String s)`

Inheritance

`java.lang.Object` → `java.lang.Throwable` → `java.lang.Exception` → `LocationException`

Methods inherited from class `java.lang.Throwable`
`fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getMessage`, `getStackTrace`, `initCause`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Description

The `LocationException` is thrown when a Location API specific error has occurred. The detailed conditions when this exception is thrown are documented in the methods that throw this exception.

Constructors

LocationException()

```
public LocationException ()
```

Constructs a `LocationException` with no detail message.

LocationException(java.lang.String)

```
public LocationException (java.lang.String s)
```

Constructs a `LocationException` with the specified detail message.

Parameters

s - the detail message

LocationListener

LocationListener — interface in package `javax.microedition.location`

```
public interface LocationListener
```

Method summary

void	locationUpdated(LocationProvider provider,Location location)
void	providerStateChanged(LocationProvider provider, int newState)

Description

The `LocationListener` represents a listener that receives events associated with a particular LocationProvider . Applications implement this interface and register it with a `LocationProvider` to obtain regular position updates.

When the listener is registered with a `LocationProvider` with some update period, the implementation shall attempt to provide updates at the defined interval. If it isn't possible to determine the location, e.g. because of the `LocationProvider` being `TEMPORARILY_UNAVAILABLE` or `OUT_OF_SERVICE` or because the update period is too frequent for the location method to provide updates, the implementation can send an update to the listener that contains an 'invalid' `Location` instance.

The implementation shall use best effort to post the location updates at the specified interval, but this timing is not guaranteed to be very exact (i.e. this is not an exact timer facility for an application).

The application is responsible for any possible synchronization needed in the listener methods.

The listener methods **must** return quickly and should not perform any extensive processing. The method calls are intended as triggers to the application. Application should do any necessary extensive processing in a separate thread and only use these methods to initiate the processing.

Methods

locationUpdated(LocationProvider , Location)

```
public void locationUpdated( LocationProvider  provider,  Location  location)
```

Called by the `LocationProvider` to which this listener is registered. This method will be called periodically according to the interval defined when registering the listener to provide updates of the current location.

Parameters

provider - the source of the event
location - the location to which the event relates, i.e. the new position

providerStateChanged(LocationProvider , int)

```
public void providerStateChanged( LocationProvider  provider, int newState)
```

Called by the `LocationProvider` to which this listener is registered if the state of the `LocationProvider` has changed.

These provider state changes are delivered to the application as soon as possible after the state of a provider changes. The timing of these events is not related to the period of the location updates.

If the application is subscribed to receive periodic location updates, it will continue to receive these regardless of the state of the `LocationProvider` . If the application wishes to stop receiving location updates for an unavailable provider, it should de-register itself from the provider.

Parameters

provider - the source of the event

newState - The new state of the `LocationProvider` . This value is one of the constants for the state defined in the `LocationProvider` class.

LocationProvider

LocationProvider — class in package `javax.microedition.location`

```
public abstract class LocationProvider extends java.lang.Object
```

Field summary

`static int`	`AVAILABLE`
`static int`	`OUT_OF_SERVICE`
`static int`	`TEMPORARILY_UNAVAILABLE`

Constructor summary

`LocationProvider()`

Method summary

`static void`	`addProximityListener(ProximityListener listener,Coordinates coordinates,float proximityRadius)`
`static LocationProvider`	`getInstance(Criteria criteria)`
`static Location`	`getLastKnownLocation()`
`Location`	`getLocation(int timeout)`
`int`	`getState()`
`static void`	`removeProximityListener(ProximityListener listener)`
`void`	`reset()`
`void`	`setLocationListener(LocationListener listener,int interval,int timeout,int maxAge)`

Inheritance

`java.lang.Object` → `LocationProvider`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

This is the starting point for applications using this API and represents a source of the location information. A `LocationProvider` represents a location-providing module, generating `Location` s.

Applications obtain `LocationProvider` instances (classes implementing the actual functionality by extending this abstract class) by calling the factory method. It is the responsibility of the implementation to return the correct `LocationProvider` -derived object.

Applications that need to specify criteria for the location provider selection, must first create a Criteria object, and pass it to the factory method. The methods that access the location related information shall throw `SecurityException` if the application does not have the relevant permission to access the location information.

Fields

AVAILABLE

```
public static final int AVAILABLE
```

Availability status code: the location provider is available.

Value: 1

OUT_OF_SERVICE

```
public static final int OUT_OF_SERVICE
```

Availability status code: the location provider is out of service. Being out of service means that the method is unavailable and the implementation is not able to expect that this situation would change in the near future. An example is when using a location method implemented in an external device and the external device is detached.

Value: 3

TEMPORARILY_UNAVAILABLE

```
public static final int TEMPORARILY_UNAVAILABLE
```

Availability status code: the location provider is temporarily unavailable. Temporary unavailability means that the method is unavailable due to reasons that can be expected to possibly change in the future and the provider to become available. An example is not being able to receive the signal because the signal used by the location method is currently being obstructed, e.g. when deep inside a building for satellite based methods. However, a very short transient obstruction of the signal should not cause the provider to toggle quickly between `TEMPORARILY_UNAVAILABLE` and `AVAILABLE` .

Value: 2

Constructors

LocationProvider()

```
protected LocationProvider ()
```

Empty constructor to help implementations and extensions. This is not intended to be used by applications. Applications should not make subclasses of this class and invoke this constructor from the subclass.

Methods

addProximityListener(ProximityListener , Coordinates , float)

```
public static void addProximityListener( ProximityListener  listener,  Coordinates
coordinates, float proximityRadius) throws javax.microedition.location.LocationException
```

Adds a `ProximityListener` for updates when proximity to the specified coordinates is detected.

If this method is called with a ProximityListener that is already registered, the registration to the specified coordinates is added in addition to the set of coordinates it has been previously registered for. A single listener can handle events for multiple sets of coordinates.

If the current location is known to be within the proximity radius of the specified coordinates, the listener shall be called immediately.

Detecting the proximity to the defined coordinates is done on a best effort basis by the implementation. Due to the limitations of the methods used to implement this, there are no guarantees that the proximity is always detected; especially in situations where the terminal briefly enters the proximity area and exits it shortly afterwards, it is possible that the implementation misses this. It is optional to provide this feature as it may not be reasonably implementable with all methods used to implement this API.

If the implementation is capable of supporting the proximity monitoring and has resources to add the new listener and coordinates to be monitored but the monitoring can not be currently done due to the current state of the method used to implement it, this method shall succeeed and the `ProximityListener.monitoringStateChanged` method of the listener shall be immediately called to notify that the monitoring is not active currently.

Parameters

`listener` - the listener to be registered
`coordinates` - the coordinates to be registered
`proximityRadius` - the radius in meters that is considered to be the threshold for being in the proximity of the specified coordinates

Throws

`IllegalArgumentException` - if the proximity radius is 0 or negative or `Float.NaN`
`NullPointerException` - if the `listener` or `coordinates` parameter is `null`
`LocationException` - if the platform does not have resources to add a new listener and coordinates to be monitored or does not support proximity monitoring at all
`SecurityException` - if the application does not have the permission to register a proximity listener
`javax.microedition.location.LocationException`

getInstance(Criteria)

```
public static  LocationProvider  getInstance( Criteria  criteria) throws
javax.microedition.location.LocationException
```

This factory method is used to get an actual `LocationProvider` implementation based on the defined criteria. The implementation chooses the `LocationProvider` so that it best fits the defined criteria, taking into account also possible implementation dependent preferences of the end user. If no concrete `LocationProvider` could be created that typically can match the defined criteria but there are other location providers not meeting the criteria that could be returned for a more relaxed criteria, `null` is returned to indicate this. The LocationException is thrown, if all supported location providers are out of service.

A `LocationProvider` instance is returned if there is a location provider meeting the criteria in either the available or temporarily unavailable state. Implementations should try to select providers in the available state before providers in temporarily unavailable state, but this can not be always guaranteed because the implementation may not always know the state correctly at this point in time. If a `LocationProvider` meeting the criteria can be supported but is currently out of service, it shall not be returned.

When this method is called with a Criteria that has all fields set to the default values (i.e. the least restrictive criteria possible), the implementation shall return a `LocationProvider` if there is any provider that is not in the out of service state. Passing `null` as the parameter is equal to passing a `Criteria` that has all fields set to the default values, i.e. the least restrictive set of criteria.

This method only makes the selection of the provider based on the criteria and is intended to return it quickly to the application. Any possible initialization of the provider is done at an implementation dependent time and **must not** block the call to this method.

This method may, depending on the implementation, return the same `LocationProvider` instance as has been returned previously from this method to the calling application, if the same instance can be used to fulfil both defined criteria. Note that there can be only one LocationListener associated with a `LocationProvider` instance.

Parameters

`criteria` - the criteria for provider selection or `null` to indicate the least restrictive criteria with default values

Returns

a `LocationProvider` meeting the defined criteria or `null` if a `LocationProvider` that meets the defined criteria can't be returned but there are other supported available or temporarily unavailable providers that do not meet the criteria.

Throws

`LocationException` - if all `LocationProvider`s are currently out of service
`javax.microedition.location.LocationException`

getLastKnownLocation()

```
public static  Location  getLastKnownLocation()
```

Returns the last known location that the implementation has. This is the best estimate that the implementation has for the previously known location.

Applications can use this method to obtain the last known location and check the timestamp and other fields to determine if this is recent enough and good enough for the application to use without needing to make a new request for the current location.

Returns

a `Location` object, `null` is returned if the implementation does not have any previous location information.

Throws

`SecurityException` - if the calling application does not have a permission to query the location information

getLocation(int)

```
public  Location  getLocation(int timeout) throws
javax.microedition.location.LocationException, java.lang.InterruptedException
```

Retrieves a Location with the constraints given by the Criteria associated with this class. If no result could be retrieved, a LocationException is thrown. If the location can not be determined within the timeout period specified in the parameter, the method shall throw a `LocationException` .

If the provider is temporarily unavailable, the implementation shall wait and try to obtain the location until the timeout expires. If the provider is out of service, then the `LocationException` is thrown immediately.

Note that the individual `Location` returned might not fulfil exactly the criteria used for selecting this `LocationProvider` . The `Criteria` is used to select a location provider that typically is able to meet the defined criteria, but not necessarily for every individual location measurement.

Parameters

`timeout` - a timeout value in seconds, -1 is used to indicate that the implementation shall use its default timeout value for this provider.

Returns

a `Location` object

Throws

`IllegalArgumentException` -
`InterruptedException` - if the operation is interrupted by calling `reset()` from another thread
`LocationException` - if the location could not be retrieved or if the timeout period expired
`SecurityException` - if the calling application does not have a permission to query the location information
`javax.microedition.location.LocationException`
`java.lang.InterruptedException`

getState()

```
public int getState()
```

Returns the current state of this `LocationProvider` . The return value shall be one of the availability status code constants defined in this class.

Returns

the availability state of this `LocationProvider`

removeProximityListener(ProximityListener)

```
public static void removeProximityListener( ProximityListener  listener)
```

Removes a `ProximityListener` from the list of recipients for updates. If the specified listener is not registered or if the parameter is `null` , this method silently returns with no action.

Parameters

`listener` - the listener to remove

reset()

```
public void reset()
```

Resets the `LocationProvider` .

All pending synchronous location requests will be aborted and any blocked `LocationProvider.getLocation` method calls will terminate with `InterruptedException` .

Applications can use this method e.g. when exiting to have its threads freed from blocking synchronous operations.

setLocationListener(LocationListener , int, int, int)

```
public void setLocationListener( LocationListener  listener, int interval, int timeout,
int maxAge)
```

Adds a LocationListener for updates at the defined interval. The listener will be called with updated location at the defined interval. The listener also gets updates when the availablilty state of the `LocationProvider` changes.

Passing in -1 as the interval selects the default interval which is dependent on the used location method. Passing in 0 as the interval registers the listener to only receive provider status updates and not location updates at all.

Only one listener can be registered with each `LocationProvider` instance. Setting the listener replaces any possibly previously set listener. Setting the listener to `null` cancels the registration of any previously set listener.

The implementation shall initiate obtaining the first location result immediately when the listener is registered and provide the location to the listener as soon as it is available. Subsequent location updates will happen at the defined interval after the first one. If the specified update interval is smaller than the time it takes to obtain the first result, the listener shall receive location updates with invalid `Location` s at the defined interval until the first location result is available. Note that prior to getting the first valid location result, the `timeout` parameter has no effect. When the first valid location result is obtained, the implementation may return it to the application immediately, i.e. before the next interval is due. This implies that in the beginning when starting to obtain location results, the listener may first get updates with invalid location results at the defined interval and when the first valid location result is obtained, it may be returned to the listener as soon as it is available before the next interval is due. After the first valid location result is delivered to the application the timeout parameter is used and the next update is delivered between the time defined by the interval and time interval+timeout after the previous update.

The `timeout` parameter determines a timeout that is used if it is not possible to obtain a new location result when the update is scheduled to be provided. This timeout value indicates how many seconds the update is allowed to be provided late compared to the defined interval. If it is not possible to get a new location result (interval + timeout) seconds after the previous update, the update will be made and an invalid Location instance is returned. This is also done if the reason for the inability to obtain a new location result is due to the provider being temporarily unavailable or out of service. For example, if the `interval` is 60 seconds and the `timeout` is 10 seconds, the update must be delivered at most 70 seconds after the previous update and if no new location result is available by that time the update will be made with an invalid `Location` instance.

The `maxAge` parameter defines how old the location result is allowed to be provided when the update is made. This allows the implementation to reuse location results if it has a recent location result when the update is due to be delivered. This parameter can only be used to indicate a larger value than the normal time of obtaining a location result by a location method. The normal time of obtaining the location result means the time it takes normally to obtain the result when a request is made. If the application specifies a time value that is less than what can be realized with the used location method, the implementation shall provide as recent location results as are possible with the used location method. For example, if the `interval` is 60 seconds, the `maxAge` is 20 seconds and normal time to obtain the result is 10 seconds, the implementation would normally start obtaining the result 50 seconds after the previous update. If there is a location

result otherwise available that is more recent than 40 seconds after the previous update, then the `maxAge` setting to 20 seconds allows to return this result and not start obtaining a new one.

The requirements for the intervals hold while the application is executing. If the application environment or the device is suspended so that the application will not execute at all and then the environment is later resumed, the periodic updates **must** continue at the defined interval but there may be a shift after the suspension period compared to the original interval schedule.

Parameters

`listener` - The listener to be registered. If set to `null` the registration of any previously set listener is cancelled.
`interval` - The interval in seconds. -1 is used for the default interval of this provider. 0 is used to indicate that the application wants to receive only provider status updates and not location updates at all.
`timeout` - Timeout value in seconds, must be greater than 0. If the value is -1, the default timeout for this provider is used. Also, if the `interval` is -1 to indicate the default, the value of this parameter has no effect and the default timeout for this provider is used. If `timeout` == -1 and `interval` > 0 and the default timeout of the provider is greater than the specified interval, then the `timeout` parameter is handled as if the application had passed the same value as `timeout` as the `interval` (i.e. `timeout` is considered to be equal to the `interval`). If the `interval` is 0, this parameter has no effect.
`maxAge` - Maximum age of the returned location in seconds, must be greater than 0 or equal to -1 to indicate that the default maximum age for this provider is used. Also, if the `interval` is -1 to indicate the default, the value of this parameter has no effect and the default maximum age for this provider is used. If `maxAge` == -1 and `interval` > 0 and the default maximum age of the provider is greater than the specified interval, then the `maxAge` parameter is handled as if the application had passed the same value as `maxAge` as the `interval` (i.e. `maxAge` is considered to be equal to the `interval`). If the `interval` is 0, this parameter has no effect.

Throws

`IllegalArgumentException` -
`SecurityException` - if the calling application does not have a permission to query the location information

Orientation

Orientation — class in package `javax.microedition.location`

`public class Orientation extends java.lang.Object`

Constructor summary

`Orientation(float azimuth,boolean isMagnetic,float pitch,float roll)`

Method summary

`float`	`getCompassAzimuth()`
`static Orientation`	`getOrientation()`
`float`	`getPitch()`
`float`	`getRoll()`
`boolean`	`isOrientationMagnetic()`

Inheritance

`java.lang.Object` → `Orientation`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

The `Orientation` class represents the physical orientation of the terminal. Orientation is described by azimuth to north (the horizontal pointing direction), pitch (the vertical elevation angle) and roll (the rotation of the terminal around its own longitudinal axis).

Support for orientation information is conditionally mandatory. If the terminal has a compass hardware, the `Orientation` **must** provided. If the terminal has a 3D accelerometer hardware in addition to compass hardware, providing pitch and roll information is optional.

It is not expected that all terminals will support all of these parameters. If a terminal supports getting the `Orientation` , it **must** provide the compass azimuth information. Most commonly, this class will be used to obtain the current compass direction. Providing the pitch and roll is optional. Pitch and roll information are not meant to be used as an accelerometer data, but only to help in determining the accuracy of the azimuth value. Access to the accelerometer data should be done through other APIs, for example, through JSR 256 Sensor API.

It is up to the terminal to define its own axes, but it is generally recommended that the longitudinal axis is aligned with the bottom-to-top direction of the screen. This means that the pitch is positive when the top of the screen is up and the bottom of the screen down (when roll is zero). The roll is positive when the device is tilted clockwise looking from the direction of the bottom of the screen, i.e. when the left side of the screen is up and the right side of the screen is down (when pitch is zero).

No accuracy data is given for `Orientation` .

This class is only a container for the information. The constructor does not validate the parameters passed in but just retains the values. The `get*` methods return the values passed in the constructor. When the platform implementation returns `Orientation` objects, it **must** ensure that it only returns objects where the parameters have values set as described for their semantics in this class.

Constructors

Orientation(float, boolean, float, float)

```
public Orientation (float azimuth, boolean isMagnetic, float pitch, float roll)
```

Constructs a new `Orientation` object with the compass azimuth, pitch and roll parameters specified.

The values are expressed in degress using floating point values.

If the pitch or roll is undefined, the parameter shall be given as `Float.NaN`.

Parameters

`azimuth` - The compass azimuth relative to true or magnetic north. Valid range: [0.0, 360.0). For example, value 0.0 indicates north, 90.0 east, 180.0 south and 270.0 west.
`isMagnetic` - a boolean stating whether the compass azimuth is given as relative to the magnetic field of the Earth (= `true`) or to true north and gravity (= `false`)
`pitch` - the pitch of the terminal in degrees, valid range: [-90.0, 90.0]
`roll` - the roll of the terminal in degrees, valid range: [-180.0, 180.0)

Methods

getCompassAzimuth()

```
public float getCompassAzimuth()
```

Returns the terminal's horizontal compass azimuth in degrees relative to either magnetic or true north. The value is always in the range [0.0, 360.0) degrees. The isOrientationMagnetic() method indicates whether the returned azimuth is relative to true north or magnetic north.

Returns

the terminal's compass azimuth in degrees relative to true or magnetic north

getOrientation()

```
public static Orientation getOrientation() throws
javax.microedition.location.LocationException
```

Returns the terminal's current orientation.

Returns

an `Orientation` object containing the terminal's current orientation or `null` if the orientation can not be currently determined

Throws

`LocationException` - if the implementation does not support orientation determination
`SecurityException` - if the calling application does not have a permission to query the orientation
`javax.microedition.location.LocationException`

getPitch()

```
public float getPitch()
```

Returns the terminal's tilt in degrees defined as an angle in the vertical plane orthogonal to the ground, and through the longitudinal axis of the terminal. The value is always in the range [-90.0, 90.0] degrees. A negative value means that the top of the terminal screen is pointing towards the ground.

Returns

the terminal's pitch in degrees or `Float.NaN` if not available

getRoll()

```
public float getRoll()
```

Returns the terminal's rotation in degrees around its own longitudinal axis. The value is always in the range [-180.0, 180.0) degrees. A negative value means that the terminal is orientated anti-clockwise from its default orientation, looking from direction of the bottom of the screen.

Returns

the terminal's roll in degrees or `Float.NaN` if not available

isOrientationMagnetic()

```
public boolean isOrientationMagnetic()
```

Returns a `boolean` value that indicates whether this `Orientation` is relative to the magnetic field of the Earth or relative to true north and gravity. If this method returns `true` , the compass azimuth and pitch are relative to the magnetic field of the Earth. If this method returns `false` , the compass azimuth isrelative to true north and pitch is relative to gravity.

Returns

`true` if this `Orientation` is relative to the magnetic field of the Earth, `false` if this `Orientation` is relative to true north and gravity

ProximityListener

ProximityListener — interface in package `javax.microedition.location`

```
public interface ProximityListener
```

Method summary

`void`	`monitoringStateChanged(boolean isMonitoringActive)`
`void`	`proximityEvent(Coordinates coordinates,Location location)`

Description

This interface represents a listener to events associated with detecting proximity to some registered coordinates. Applications implement this interface and register it with a static method in LocationProvider to obtain notifications when proximity to registered coordinates is detected.

This listener is called when the terminal enters the proximity of the registered coordinates. The proximity is defined as the proximity radius around the coordinates combined with the horizontal accuracy of the current sampled location.

The listener is called only once when the terminal enters the proximity of the registered coordinates. The registration with these coordinates is canceled when the listener is called. If the application wants to be notified again about these coordinates, it must re-register the coordinates and the listener.

Methods

monitoringStateChanged(boolean)

```
public void monitoringStateChanged(boolean isMonitoringActive)
```

Called to notify that the state of the proximity monitoring has changed.

These state changes are delivered to the application as soon as possible after the state of the monitoring changes.

Regardless of the state, the `ProximityListener` remains registered until the application explicitly removes it with `LocationProvider.removeProximityListener` method or the application exits.

These state changes may be related to state changes of some location providers, but this is implementation dependent as implementations can freely choose the method used to implement this proximity monitoring.

Parameters

`isMonitoringActive` - A `boolean` indicating the new state of the proximity monitoring. `true` indicates that the proximity monitoring is active and `false` indicates that the proximity monitoring can not be done currently.

proximityEvent(Coordinates , Location)

```
public void proximityEvent( Coordinates  coordinates,  Location  location)
```

After registering this listener with the `LocationProvider` , this method will be called by the platform when the implementation detects that the current location of the terminal is within the defined proximity radius of the registered coordinates.

Parameters

`coordinates` - the registered coordinates to which proximity has been detected
`location` - the current location of the terminal

QualifiedCoordinates

QualifiedCoordinates — class in package `javax.microedition.location`

`public class QualifiedCoordinates extends Coordinates`

Constructor summary

`QualifiedCoordinates(double latitude,double longitude,float altitude, float horizontalAccuracy,float verticalAccuracy)`

Method summary

`float`	`getHorizontalAccuracy()`
`float`	`getVerticalAccuracy()`
`void`	`setHorizontalAccuracy(float horizontalAccuracy)`
`void`	`setVerticalAccuracy(float verticalAccuracy)`

Inheritance

`java.lang.Object` → `Coordinates` → `QualifiedCoordinates`

Fields inherited from `javax.microedition.location.Coordinates`
`DD_MM` , `DD_MM_SS`

Methods inherited from class `javax.microedition.location.Coordinates`
`azimuthTo` , `convert` , `convert` , `distance` , `getAltitude` , `getLatitude` , `getLongitude` , `setAltitude` , `setLatitude` , `setLongitude`

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Description

The `QualifiedCoordinates` class represents coordinates as latitude-longitude-altitude values that are associated with an accuracy value.

Constructors

QualifiedCoordinates(double, double, float, float, float)

`public QualifiedCoordinates (double latitude, double longitude, float altitude, float horizontalAccuracy, float verticalAccuracy)`

Constructs a new `QualifiedCoordinates` object with the values specified. The latitude and longitude parameters are expressed in degrees using floating point values. The degrees are in decimal values (rather than minutes/seconds).

The coordinate values always apply to the WGS84 datum.

The `Float.NaN` value can be used for altitude to indicate that altitude is not known.

Parameters

`latitude` - the latitude of the location, valid range: [-90.0, 90.0]
`longitude` - the longitude of the location, valid range: [-180.0, 180.0)
`altitude` - The altitude of the location in meters, defined as height above WGS84 ellipsoid. `Float.NaN` can be used to indicate that altitude is not known.

horizontalAccuracy - The horizontal accuracy of this location result in meters. `Float.NaN` can be used to indicate that the accuracy is not known. Must be greater or equal to 0.
verticalAccuracy - The vertical accuracy of this location result in meters. `Float.NaN` can be used to indicate that the accuracy is not known. Must be greater or equal to 0.

Throws

`IllegalArgumentException` - if an input parameter is out of the valid range

Methods

getHorizontalAccuracy()

```
public float getHorizontalAccuracy()
```

Returns the horizontal accuracy of the location in meters (1-sigma standard deviation). A value of `Float.NaN` means the horizontal accuracy could not be determined.

The horizontal accuracy is the RMS (root mean square) of east accuracy (latitudinal error in meters, 1-sigma standard deviation), north accuracy (longitudinal error in meters, 1-sigma).

Returns

the horizontal accuracy in meters, `Float.NaN` if this is not known

getVerticalAccuracy()

```
public float getVerticalAccuracy()
```

Returns the accuracy of the location in meters in vertical direction (orthogonal to ellipsoid surface, 1-sigma standard deviation). A value of `Float.NaN` means the vertical accuracy could not be determined.

Returns

the vertical accuracy in meters, `Float.NaN` if this is not known.

setHorizontalAccuracy(float)

```
public void setHorizontalAccuracy(float horizontalAccuracy)
```

Sets the horizontal accuracy of the location in meters (1-sigma standard deviation). A value of `Float.NaN` means the horizontal accuracy could not be determined.

The horizontal accuracy is the RMS (root mean square) of east accuracy (latitudinal error in meters, 1-sigma standard deviation), north accuracy (longitudinal error in meters, 1-sigma).

Parameters

horizontalAccuracy - The horizontal accuracy of this location result in meters. `Float.NaN` means the horizontal accuracy could not be determined. Must be greater or equal to 0.

Throws

`IllegalArgumentException` - if the parameter is less than 0

setVerticalAccuracy(float)

```
public void setVerticalAccuracy(float verticalAccuracy)
```

Sets the accuracy of the location in meters in vertical direction (orthogonal to ellipsoid surface, 1-sigma standard deviation). A value of `Float.NaN` means the vertical accuracy could not be determined.

Parameters

verticalAccuracy - The vertical accuracy of this location result in meters. `Float.NaN` means the horizontal accuracy could not be determined. Must be greater or equal to 0.

Throws

`IllegalArgumentException` - if the parameter is less than 0

Constant field values

Package javax.microedition.location

AddressInfo

BUILDING_FLOOR	11
BUILDING_NAME	10
BUILDING_ROOM	12
BUILDING_ZONE	13
CITY	4
COUNTRY	7
COUNTRY_CODE	8
COUNTY	5
CROSSING1	14
CROSSING2	15
DISTRICT	9
EXTENSION	1
PHONE_NUMBER	17
POSTAL_CODE	3
STATE	6
STREET	2
URL	16

Coordinates

DD_MM	2
DD_MM_SS	1

Criteria

NO_REQUIREMENT	0
POWER_USAGE_HIGH	3
POWER_USAGE_LOW	1
POWER_USAGE_MEDIUM	2

Location

MTA_ASSISTED	262144
MTA_UNASSISTED	524288
MTE_ANGLEOFARRIVAL	32
MTE_CELLID	8
MTE_SATELLITE	1
MTE_SHORTRANGE	16
MTE_TIMEDIFFERENCE	2
MTE_TIMEOFARRIVAL	4
MTY_NETWORKBASED	131072

MTY_TERMINALBASED	65536

LocationProvider

AVAILABLE	1
OUT_OF_SERVICE	3
TEMPORARILY_UNAVAILABLE	2

Appendix A. LocationPermission

This appendix provides the skeleton of the `LocationPermission` class, containing the API signatures and documentation.

```
  package javax.microedition.location;

import java.security.Permission;

/**
 * <p>This class controls the access to the protected functionality in Location
 * API. The following table defines the protected methods and the permission
 * name and action parameters needed to access these methods.</p>
 *
 * <table border='1'>
 *  <caption><code>name</code> and <code>actions</code> parameters in
 *           LocationPermission</caption>
 *  <thead>
 *      <tr>
 *          <th>API call</th>
 *          <th>Name parameter in <code>LocationPermission</code></th>
 *          <th>Actions parameters in <code>LocationPermission</code></th>
 *      </tr>
 *  </thead>
 *  <tbody>
 *      <tr>
 *          <td><code>LocationProvider.getLocation()</code>,
 *              <code>LocationProvider.setLocationListener()</code>,
 *              <code>LocationProvider.getLastKnownLocation()</code></td>
 *          <td>"location"</td>
 *          <td>"location"</td>
 *      </tr>
 *      <tr>
 *          <td><code>LocationProvider.addProximityListener()</code></td>
 *          <td>"proximity_listener"</td>
 *          <td>"proximity"</td>
 *      </tr>
 *      <tr>
 *          <td><code>Orientation.getOrientation()</code></td>
 *          <td>"orientation"</td>
 *          <td>"orientation"</td>
 *      </tr>
 *      <tr>
 *          <td><code>LandmarkStore.getInstance()</code>,
 *              <code>LandmarkStore.listLandmarkStores()</code></td>
 *          <td>"landmark_store"</td>
 *          <td>"read"</td>
 *      </tr>
 *      <tr>
 *          <td><code>LandmarkStore.addLandmark()</code>,
 *              <code>LandmarkStore.deleteLandmark()</code>,
 *              <code>LandmarkStore.removeLandmarkFromCategory()</code>,
 *              <code>LandmarkStore.updateLandmark()</code></td>
 *          <td>"landmark_store"</td>
 *          <td>"write"</td>
 *      </tr>
 *      <tr>
 *          <td><code>LandmarkStore.addCategory()</code>,
 *              <code>LandmarkStore.deleteCategory()</code></td>
 *          <td>"landmark_store"</td>
 *          <td>"category"</td>
 *      </tr>
 *      <tr>
```

```
 *          <td><code>LandmarkStore.createLandmarkStore()</code>,
 *              <code>LandmarkStore.deleteLandmarkStore()</code></td>
 *          <td>"landmark_store"</td>
 *          <td>"management"</td>
 *      </tr>
 *  </tbody>
 * </table>
 *
 * @author Nokia Corporation
 * @version 1.1
 */
public final class LocationPermission extends Permission {

    /**
     * <p>Creates a new <code>LocationPermission</code> object with the specified
     * name and actions. The <code>name</code> parameter is the class to which
     * the permissions apply. Possible <code>name</code> values are "location",
     * "orientation", "proximity_listener" and "landmark_store". Parameter
     * <code>actions</code> contains a comma separated list of actions granted to
     * the class specified by the <code>name</code> parameter. Possible action
     * values are "location", "orientation", "proximity", "read", "write", "category"
     * and "management". The possible combinations between names and actions
     * are defined in the class description. If the same action is given in the
     * <code>actions</code> multiple times, additional entries are ignored.</p>
     *
     * @param name the functionality to which the permission applies
     * @param actions a comma separated list of actions
     * @throws NullPointerException if <code>name</code> or <code>actions</code>
     *                  is <code>null</code>
     * @throws IllegalArgumentException if <code>name</code> and <code>actions</code>
     *                 combination is not according to table in the class description
     */
    public LocationPermission(String name, String actions) {
        super(name);
    }

    /**
     * <p>Checks if this <code>LocationPermission</code> object "implies" the
     * specified permission.</p>
     *
     * <p>The method returns <code>true</code> if:</p>
     *
     * <ul>
     *  <li><code>p</code> is an instance of <code>LocationPermission</code>,</li>
     *  <li><code>p</code>'s name is equal to the name of this object, and</li>
     *  <li><code>p</code>'s actions are a proper subset of this object's actions</li>
     * </ul>
     *
     * @param p the permission to check against
     * @return <code>true</code> if the specified permission is implied by this
     *              object, <code>false</code> if not
     */
    public boolean implies(Permission p) {
        return false;
    }

    /**
     * <p>Checks two <code>LocationPermission</code> objects for equality. Checks
     * that <code>obj</code> is a <code>LocationPermission</code> and has the same
     * name and actions as this object.</p>
     *
     * @param obj the object we are testing for equality with this object
     * @return <code>true</code> if <code>obj</code> is equal to this object,
     *              else <code>false</code>
```

```
     */
    public boolean equals(Object obj) {
        return false;
    }

    /**
     * <p>Returns the hash code value for this object.</p>
     *
     * @return this object's hash code
     */
    public int hashCode() {
        return 0;
    }

    /**
     * <p>Returns the canonical string representation of the current actions.
     * Actions are separated with a comma. The actions for different names are
     * defined in the table in the class description. For the
     * <code>"landmark_store"</code> name the actions
     * <span class="keyword">must</span> be returned in the following order:
     * <code>read</code>, <code>write</code>, <code>category</code>,
     * <code>management</code>.</p>
     *
     * <p>If this <code>LocationPermission</code> allows reading and writing
     * landmarks but not management of categories of landmark stores, a call to
     * <code>getActions</code> will return the string "read,write".</p>
     *
     * @return the canonical string representation of the actions
     */
    public String getActions() {
        return null;
    }

}
```

Index

I

L

M