SiRFstudio API

version 0.9

SiRF Technologies, Ltd.

Package com.sirf.microedition.location

com.sirf.microedition.location
Class AddressInfo

```
java.lang.Object
  |
  +-com.sirf.microedition.location.AddressInfo
```

public class **AddressInfo**
extends Object

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		

Field Summary

`public static final`	`BUILDING_FLOOR` Address field denoting a building floor. Value: **11**
`public static final`	`BUILDING_NAME` Address field denoting a building name. Value: **10**
`public static final`	`BUILDING_ROOM` Address field denoting a building room. Value: **12**
`public static final`	`BUILDING_ZONE` Address field denoting a building zone Value: **13**
`public static final`	`CITY` Address field denoting town or city name. Value: **4**
`public static final`	`COUNTRY` Address field denoting country. Value: **7**
`public static final`	`COUNTRY_CODE` Address field denoting country as a two-letter ISO 3166-1 code. Value: **8**
`public static final`	`COUNTY` Address field denoting a county, which is an entity between a state and a city Value: **5**
`public static final`	`CROSSING1` Address field denoting a street in a crossing. Value: **14**
`public static final`	`CROSSING2` Address field denoting a street in a crossing. Value: **15**
`public static final`	`DISTRICT` Address field denoting a municipal district. Value: **9**
`public static final`	`EXTENSION` Address field denoting address extension, e.g. Value: **1**
`public static final`	`PHONE_NUMBER` Address field denoting a phone number for this place. Value: **17**
`public static final`	`POSTAL_CODE` Address field denoting zip or postal code. Value: **3**

`public static final`	`STATE` Address field denoting state or province. Value: **6**
`public static final`	`STREET` Address field denoting street name and number. Value: **2**
`public static final`	`URL` Address field denoting a URL for this place. Value: **16**

Constructor Summary

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

Method Summary

`String`	`getField(int field)` Returns the value of an address field.
`void`	`setField(int field, String value)` Sets the value of an address field.
`String`	`toString()`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

EXTENSION

`public static final int` **`EXTENSION`**

Address field denoting address extension, e.g. flat number.
Constant value: **1**

STREET

`public static final int` **`STREET`**

Address field denoting street name and number.
Constant value: **2**

POSTAL_CODE

`public static final int` **`POSTAL_CODE`**

Address field denoting zip or postal code.
Constant value: **3**

CITY

`public static final int` **`CITY`**

Address field denoting town or city name.
Constant value: **`4`**

COUNTY

`public static final int` **`COUNTY`**

Address field denoting a county, which is an entity between a state and a city
Constant value: **`5`**

STATE

`public static final int` **`STATE`**

Address field denoting state or province.
Constant value: **`6`**

COUNTRY

`public static final int` **`COUNTRY`**

Address field denoting country.
Constant value: **`7`**

COUNTRY_CODE

`public static final int` **`COUNTRY_CODE`**

Address field denoting country as a two-letter ISO 3166-1 code.
Constant value: **`8`**

DISTRICT

`public static final int` **`DISTRICT`**

Address field denoting a municipal district.
Constant value: **`9`**

BUILDING_NAME

`public static final int` **`BUILDING_NAME`**

Address field denoting a building name.
Constant value: **`10`**

BUILDING_FLOOR

`public static final int` **`BUILDING_FLOOR`**

Address field denoting a building floor.
Constant value: **`11`**

(continued on next page)

(continued from last page)

BUILDING_ROOM

`public static final int **BUILDING_ROOM**`

Address field denoting a building room.
Constant value: **12**

BUILDING_ZONE

`public static final int **BUILDING_ZONE**`

Address field denoting a building zone
Constant value: **13**

CROSSING1

`public static final int **CROSSING1**`

Address field denoting a street in a crossing.
Constant value: **14**

CROSSING2

`public static final int **CROSSING2**`

Address field denoting a street in a crossing.
Constant value: **15**

URL

`public static final int **URL**`

Address field denoting a URL for this place.
Constant value: **16**

PHONE_NUMBER

`public static final int **PHONE_NUMBER**`

Address field denoting a phone number for this place.
Constant value: **17**

Constructors

AddressInfo

`public **AddressInfo**()`

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

Methods

getField

`public String **getField**(int field)`

(continued on next page)

(continued from last page)

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

See Also:
`setField(int, String)`

setField

```
public void setField(int field,
         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

See Also:
`getField(int)`

toString

```
public String toString()
```

com.sirf.microedition.location
Class Coordinates

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Coordinates
```

Direct Known Subclasses:
QualifiedCoordinates

---

public class **Coordinates**
extends Object

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.

Field Summary

`public static final`	`DD_MM` Identifier for `String` coordinate representation Degrees, Minutes, decimal fractions of a minute Value: **2**
`public static final`	`DD_MM_SS` Identifier for `String` coordinate representation Degrees, Minutes, Seconds and decimal fractions of a second Value: **1**

Constructor Summary

`public`	`Coordinates(double latitude, double longitude, float altitude)` Constructs a new `Coordinates` object with the values specified.

Method Summary

`float`	`azimuthTo(Coordinates to)` Calculates the azimuth between the two points according to the ellipsoid model of WGS84.
`static String`	`convert(double coordinate, int outputType)` Converts a `double` representation of a coordinate with decimal degrees into a `String` representation.
`static double`	`convert(String coordinate)` Converts a `String` representation of a coordinate into the `double` representation as used in this API.
`float`	`distance(Coordinates to)` Calculates the geodetic distance between the two points according to the ellipsoid model of WGS84.

`float`	`getAltitude()` Returns the altitude component of this coordinate.
`double`	`getLatitude()` Returns the latitude component of this coordinate.
`double`	`getLongitude()` Returns the longitude component of this coordinate.
`void`	`setAltitude(float altitude)` Sets the geodetic altitude for this point.
`void`	`setLatitude(double latitude)` Sets the geodetic latitude for this point.
`void`	`setLongitude(double longitude)` Sets the geodetic longitude for this point.
`String`	`toString()`

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

Fields

DD_MM_SS

```
public static final int DD_MM_SS
```

Identifier for `String` coordinate representation Degrees, Minutes, Seconds and decimal fractions of a second
Constant value: **1**

DD_MM

```
public static final int DD_MM
```

Identifier for `String` coordinate representation Degrees, Minutes, decimal fractions of a minute
Constant value: **2**

Constructors

Coordinates

```
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.

(continued from last page)

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

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

See Also:

`setLatitude(double)`

---

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

See Also:

`setLongitude(double)`

---

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

See Also:

`setAltitude(float)`

setAltitude

`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

See Also:

`getAltitude()`

setLatitude

`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

See Also:

`getLatitude()`

setLongitude

`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

See Also:

`getLongitude()`

convert

`public static double **convert**(String coordinate)`

(continued from last page)

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`

convert

```
public static 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`

See Also:

`convert(String)`

azimuthTo

```
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`

distance

```
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:

(continued from last page)

`to` - the `Coordinates` of the destination

Returns:
the distance to the destination in meters

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

toString

```
public String toString()
```

com.sirf.microedition.location
Class Criteria

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Criteria
```

public class **Criteria**
extends Object

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.

SiRF Implementation Mode Chart

Horizontal Accuracy	Vertical Accuracy	Cost	Power Consumption	Resulting Mode
Required	Required	Allowed	Low, Medium,High or No Requirement	MS_BASED
Not Required	Not Required	Allowed	Low, Medium,High or No Requirement	MS_BASED

Field Summary

public static final	NO_REQUIREMENT Constant indicating no requirements for the parameter. Value: **0**
public static final	POWER_USAGE_HIGH Level indicating high power consumption allowed. Value: **3**
public static final	POWER_USAGE_LOW Level indicating only low power consumption allowed. Value: **1**
public static final	POWER_USAGE_MEDIUM Level indicating average power consumption allowed. Value: **2**

Constructor Summary

public	Criteria() Constructs a `Criteria` object.

Method Summary

int	getHorizontalAccuracy() Returns the horizontal accuracy value set in this `Criteria`.
int	getPreferredPowerConsumption() Returns the preferred power consumption.
int	getPreferredResponseTime() Returns the preferred maximum response time.
int	getVerticalAccuracy() Returns the vertical accuracy value set in this `Criteria`.
boolean	isAddressInfoRequired() Returns whether the location provider should be able to determine textual address information.
boolean	isAllowedToCost() Returns the preferred cost setting.
boolean	isAltitudeRequired() Returns whether the location provider should be able to determine altitude.
boolean	isSpeedAndCourseRequired() Returns whether the location provider should be able to determine speed and course.
void	setAddressInfoRequired(boolean addressInfoRequired) Sets whether the location provider should be able to determine textual address information.
void	setAltitudeRequired(boolean altitudeRequired) Sets whether the location provider should be able to determine altitude.
void	setCostAllowed(boolean costAllowed) Sets the preferred cost setting.

void	`setHorizontalAccuracy(int accuracy)` Sets the desired horizontal accuracy preference.
void	`setPreferredPowerConsumption(int level)` Sets the preferred maximum level of power consumption.
void	`setPreferredResponseTime(int time)` Sets the desired maximum response time preference.
void	`setSpeedAndCourseRequired(boolean speedAndCourseRequired)` Sets whether the location provider should be able to determine speed and course.
void	`setVerticalAccuracy(int accuracy)` Sets the desired vertical accuracy preference.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

NO_REQUIREMENT

`public static final int` **`NO_REQUIREMENT`**

Constant indicating no requirements for the parameter.
Constant value: **`0`**

POWER_USAGE_LOW

`public static final int` **`POWER_USAGE_LOW`**

Level indicating only low power consumption allowed.
Constant value: **`1`**

POWER_USAGE_MEDIUM

`public static final int` **`POWER_USAGE_MEDIUM`**

Level indicating average power consumption allowed.
Constant value: **`2`**

POWER_USAGE_HIGH

`public static final int` **`POWER_USAGE_HIGH`**

Level indicating high power consumption allowed.
Constant value: **`3`**

Constructors

Criteria

`public` **`Criteria`**`()`

(continued from last page)

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

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`.

See Also:
`setPreferredPowerConsumption(int)`

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

See Also:
`setCostAllowed(boolean)`

getVerticalAccuracy

`public int getVerticalAccuracy()`

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

Returns:
the accuracy in meters

See Also:
`setVerticalAccuracy(int)`

getHorizontalAccuracy

`public int getHorizontalAccuracy()`

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

Returns:
the horizontal accuracy in meters

See Also:
`setHorizontalAccuracy(int)`

getPreferredResponseTime

`public int getPreferredResponseTime()`

Returns the preferred maximum response time.

(continued from last page)

Returns:
the maximum response time in milliseconds

See Also:
`setPreferredResponseTime(int)`

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.

See Also:
`setSpeedAndCourseRequired(boolean)`

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.

See Also:
`setAltitudeRequired(boolean)`

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.

See Also:
`setAddressInfoRequired(boolean)`

setHorizontalAccuracy

`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

See Also:
`getHorizontalAccuracy()`

setVerticalAccuracy

`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

See Also:
`getVerticalAccuracy()`

setPreferredResponseTime

`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

See Also:
`getPreferredResponseTime()`

setPreferredPowerConsumption

`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`.

See Also:
`getPreferredPowerConsumption()`

setCostAllowed

`public void setCostAllowed(boolean costAllowed)`

(continued from last page)

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

See Also:
`isAllowedToCost()`

setSpeedAndCourseRequired

`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.

See Also:
`isSpeedAndCourseRequired`

setAltitudeRequired

`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.

See Also:
`isAltitudeRequired`

setAddressInfoRequired

`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.

See Also:

(continued from last page)

`isAddressInfoRequired`

com.sirf.microedition.location
Class Landmark

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Landmark
```

public class **Landmark**
extends Object

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`, `AddressInfo` and timestamp of the last change.

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. So if an application changes the `QualifiedCoordinates` of the landmark and wants to store those changes, it **must** explicitly call `LandmarkStore.updateLandmark()` method.

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. These fields are `identifier`, `author`, `extraInfo` and `timestamp`. `identifier` is unique inside a landmark store and allows the application developers refer to specified landmarks even though the name of the landmark has changed. The `author` indicates the provider or developer of the landmark and it **should** be a reverse domain name. With the `ExtraInfo` field the application developer is able to provide additional information about the landmark. This information can be, for example, speed or heading of the landmark. `timestamp` field is added to better track when the data in a landmark has been changed.

Constructor Summary	
`public`	`Landmark(String name, String description, QualifiedCoordinates coordinates, AddressInfo addressInfo)` Constructs a new `Landmark` object with the values specified.
`public`	`Landmark(String name, String description, QualifiedCoordinates coordinates, AddressInfo addressInfo, int identifier, String author, String extraInfo)` Constructs a new `Landmark` object with the values specified.

Method Summary	
`boolean`	`equals(Object obj)` Compares given Landmark and this instance for equality.
`AddressInfo`	`getAddressInfo()` Gets the `AddressInfo` of the landmark.
`String`	`getAuthor()` Returns the author of the landmark.

String	getDescription() Gets the landmark description.
String	getExtraInfo() Gets the extra information of the landmark.
int	getIdentifier() Returns the identifier of the landmark
String	getName() Gets the landmark name.
QualifiedCoordinates	getQualifiedCoordinates() Gets the `QualifiedCoordinates` of the landmark.
long	getTimestamp() Gets the timestamp of the landmark.
void	setAddressInfo(AddressInfo addressInfo) Sets the `AddressInfo` of the landmark.
void	setAuthor(String author) Sets the author or the provider of the landmark data
void	setDescription(String description) Sets the description of the landmark.
void	setExtraInfo(String extraInfo) Sets the additional information to the landmark.
void	setIdentifier(int identifier) Sets the unique identifier of the landmark.
void	setName(String name) Sets the name of the landmark.
void	setQualifiedCoordinates(QualifiedCoordinates coordinates) Sets the `QualifiedCoordinates` of the landmark.
String	toString()

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

Constructors

Landmark

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

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

(continued from last page)

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`

Landmark

```
public Landmark(String name,
                String description,
                QualifiedCoordinates coordinates,
                AddressInfo addressInfo,
                int identifier,
                String author,
                String extraInfo)
```

Constructs a new `Landmark` object with the values specified. `identifier` is the unique identifier of the landmark. The `extraInfo` is a comma separated list of additional information items. This information may not necessarily be visible to the user.

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
`identifier` - the unique idenfitier of the landmark // * @param radius the coverage radius of the landmark in meters
`author` - the provider of the landmark data, may be `null`
`extraInfo` - the additional information about the landmark, may be `null` if not available

Throws:

`java.lang.NullPointerException` - if the `name` is `null`
`java.lang.IllegalArgumentException` - if `identifier` 0

Methods

getName

```
public String getName()
```

Gets the landmark name.

Returns:

the name of the landmark

See Also:

`setName(String)`

---

getDescription

```
public String getDescription()
```

Gets the landmark description.

Returns:

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

See Also:

(continued from last page)

`setDescription(String)`

---

getQualifiedCoordinates

`public QualifiedCoordinates **getQualifiedCoordinates**()`

Gets the `QualifiedCoordinates` of the landmark.

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

See Also:
`setQualifiedCoordinates(QualifiedCoordinates)`

---

getAddressInfo

`public AddressInfo **getAddressInfo**()`

Gets the `AddressInfo` of the landmark.

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

See Also:
`setAddressInfo(AddressInfo)`

---

getIdentifier

`public int **getIdentifier**()`

Returns the identifier of the landmark

Returns:
the landmark's identifier, -1 if identifier has not been set.

---

getAuthor

`public String **getAuthor**()`

Returns the author of the landmark.

Returns:
the landmark's author or `null` if author has not been set

---

getExtraInfo

`public String **getExtraInfo**()`

Gets the extra information of the landmark. The returned `String` contains a comma separated list of additional landmark information items set by the application in the constructor or with the `setExtraInfo` method.

Returns:
the additional information, `null` if not available

See Also:
`setExtraInfo(String)`

(continued from last page)

getTimestamp

`public long getTimestamp()`

Gets the timestamp of the landmark. Timestamp indicates the time when the landmark information was last modified. The timestamp is updated by the API implementation. 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:
the time of the last update on the landmark data

setName

`public void setName(String name)`

Sets the name of the landmark.

Parameters:
`name` - name for the landmark

Throws:
`NullPointerException` - if the parameter is `null`

See Also:
`getName()`

setDescription

`public void setDescription(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

See Also:
`getDescription()`

setQualifiedCoordinates

`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 qualified coordinates are not available

See Also:
`getQualifiedCoordinates()`

setAddressInfo

`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 address information is not available

(continued from last page)

See Also:
`getAddressInfo()`

setIdentifier

```
public void setIdentifier(int identifier)
  throws IllegalArgumentException
```

Sets the unique identifier of the landmark.

Parameters:
`identifier` - the landmark's identifier

Throws:
`IllegalArgumentException` - if identifier <0

setAuthor

```
public void setAuthor(String author)
```

Sets the author or the provider of the landmark data

Parameters:
`author` - landmark's author

setExtraInfo

```
public void setExtraInfo(String extraInfo)
```

Sets the additional information to the landmark. This information may be, for example, speed or heading at the landmark. The additional information is a comma separated list of additional items.

Parameters:
`extraInfo` - additional information for the landmark, `null` may be passed in to indicate that additional information is not available

equals

```
public boolean equals(Object obj)
```

Compares given Landmark and this instance for equality. Two Landmark instances are considered to be equal if they have the following fields the same:

- Name
- Identifier, if set (<>-1)
- If qualified coords are set in both:
- Latitude
- Longitude

.

toString

```
public String toString()
```

com.sirf.microedition.location
Class LandmarkException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-com.sirf.microedition.location.LandmarkException
```

All Implemented Interfaces:
Serializable

---

public class **LandmarkException**
extends Exception

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

Constructor Summary

`public`	`LandmarkException()` Constructs a `LandmarkException` with no detail message.
`public`	`LandmarkException(String s)` Constructs a `LandmarkException` with the specified detail message.

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, toString, wait, wait, wait`

Constructors

LandmarkException

`public` **`LandmarkException`**`()`

Constructs a `LandmarkException` with no detail message.

LandmarkException

`public` **`LandmarkException`**`(String s)`

Constructs a `LandmarkException` with the specified detail message.

Parameters:
`s` - the detailed message

com.sirf.microedition.location
Class LandmarkStore

```
java.lang.Object
   |
   +-com.sirf.microedition.location.LandmarkStore
```

public class **LandmarkStore**
extends Object

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:

- U+0000...U+001F control characters
- U+005C `'\'`
- U+002F `'/'`
- U+003A `':'`
- U+002A `'*'`
- U+003F `'?'`
- U+0022 `'"'`
- U+003C `'<'`
- U+003E `'>'`
- U+007C `'|'`
- U+007F...U+009F control characters
- U+FEFF Byte-order-mark
- U+FFF0...U+FFFF

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 a 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. Appendix A defines the permissions (names etc.) for security framework in MIDP 2.0 and later versions.

New features in Location API 2.0

Global landmark categories

Location API [JSR179] does not define any landmark categories, but they are all left to the application to decide. This approach makes the landmark category names hard to localize. In Location API 2.0 a set of global landmark categories are defined. These global categories are also localized to enable better user experience. And since all devices implementing Location API 2.0 have these same categories, landmarks can be imported and exported into same global category in different devices. The user or application is still able to create their own landmark categories, which may be subcategories for some global categories.

More search capabilities

As Location API 2.0 provides capabilities for importing landmarks the number of landmarks in the landmark store may grow quite much. Therefore new and more advanced search mechanism for the landmarks is needed. The Location API 2.0 provides new search methods to search all categories for a specified landmark or search with incomplete names and wildcards. And since the Location API 2.0 adds a unique identifier for the landmarks inside a landmark store, method to search landmarks based on the identifier is also provided.

Landmark store listener

Since landmark store is shared between all Java applications and also possibly with the native applications, the content of the landmark store may be changed during the application runtime by some other application. This may cause some landmarks to appear or disappear from the landmark store. To address this problem, the Location API 2.0 adds a mechanism for the applications to be notified about the changes in the landmark store. By listening to the changes the application is able to prepare for the possible changes in the landmark store.

Method Summary	
`void`	`addCategory(String categoryName)` Adds a category to this `LandmarkStore`.
`void`	`addLandmark(Landmark landmark, String category)` Adds a landmark to the specified category in the landmark store.
`void`	`addLandmarkStoreListener(LandmarkStoreListener listener)` Adds a `LandmarkStoreListener` that is notified when there are changes in the specified landmark store.

static void	createLandmarkStore(String storeName) Creates a new landmark store with a specified name.
void	deleteCategory(String categoryName) Removes a category from this LandmarkStore.
void	deleteLandmark(Landmark landmark) Deletes a landmark from this LandmarkStore.
static void	deleteLandmarkStore(String storeName) Delete a landmark store with a specified name.
Enumeration	getCategories() Returns the category names that are defined in this LandmarkStore.
Enumeration	getCategoriesOfLandmark(Landmark landmark) Returns the landmark categories for the specified landmark.
static LandmarkStore	getInstance(String storeName) Gets a LandmarkStore instance for storing, deleting and retrieving landmarks.
Landmark	getLandmark(int identifier) Returns the Landmark object for the given identifier, null if the LandmarkStore does not contain matching landmark.
Enumeration	getLandmarks() Lists all landmarks stored in the store.
Enumeration	getLandmarks(String category, double minLatitude, double maxLatitude, double minLongitude, double maxLongitude) 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.
Enumeration	getLandmarks(String category, String name) Gets the landmarks from the storage where the category and/or name matches the given parameters.
int	getNumberOfLandmarks(String category) This convenience method returns the number of landmarks in the given category.
static String[]	listLandmarkStores() Lists the names of all the available landmark stores.
void	removeLandmarkFromAllCategories(Landmark landmark) Removes all the categories associated to the specified landmark.
void	removeLandmarkFromCategory(Landmark lm, String category) Removes the named landmark from the specified category.
void	removeLandmarkStoreListener(LandmarkStoreListener listener) Removes the specified listener from receiving notifications about changes in the landmark store.
void	updateLandmark(Landmark landmark) Updates the information about a landmark.

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

Methods

getInstance

```
public static LandmarkStore getInstance(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 SiRFStudio implementation notes: - The names of LandmarkStores are case sensitive. - If this method is called several times with the same name, different instances are returned, pointing to same databases. The operations on databases are atomic. However, since there is no mechanism notify e.g. about a deletion of a LandmarkStore, the list of LandmarkStores might get obsolete. It is the responsibility of the application to make sure it is synchronizing on the different method calls between Threads.

createLandmarkStore

```
public static void createLandmarkStore(String storeName)
  throws IOException,
         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`

(continued from last page)

`IllegalArgumentException` - if the name is too long or if a landmark store with the specified name already exists or if the name contains characters that are not supported by the API implementation
`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 SiRFStudio implementation notes: SiRFStudio implementation does not support multiple LandmarkStores with same name. Also, the name of LandmarkStore cannot be of zero length.

deleteLandmarkStore

```
public static void deleteLandmarkStore(String storeName)
  throws IOException,
         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 application does not have permissions to delete a landmark store
`LandmarkException` - if the implementation does not support deleting landmark stores

listLandmarkStores

```
public static String[] listLandmarkStores()
  throws 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

addLandmark

```
public void addLandmark(Landmark landmark,
         String category)
  throws IOException
```

(continued from last page)

Adds a landmark to the specified category 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 landmark's name field **must not** be truncated. Every implementation **must** 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, an `IllegalArgumentException` is thrown.

When the landmark store is empty, every implementation is required to be able to store a landmark where each `String` field (including landmark's description and all fields in the `AddressInfo` associated with the landmark) 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` is an instance that does not belong to this `LandmarkStore` and there already is a `Landmark` with the same identifier in the `LandmarkStore` an `IllegalArgumentException` is thrown.

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`.

If a landmark is to be added to more than one category, the additions **must** be done one by one with this method.

If there is a `LandmarkStoreListener` assigned for this landmark store, the API implementation **must** send `LandmarkStoreListener.landmarkAdded` notification to registered applications before this method returns.

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` or if the `LandmarkStore` already contains a different `Landmark` with the same identifier

`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

getLandmarks

```
public Enumeration getLandmarks(String category,
         String name)
  throws IOException
```

Gets the landmarks from the storage where the category and/or name matches the given parameters. The name may be incomplete or contain wildcards. `?` character can be used as a wildcard that matches any character. `*` character is a wild card that matches 0 or more characters. API implementations **must** support wild cards.

For example method call `getLandmarks(null, fooba?)` will match with landmark names like `"foobar"` and `"foobaz"`. Method call `getLandmarks(null, f*bar)` will match with landmark names like `"foobar"`, `"fooobar"` and `"fbar"`.

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

(continued from last page)

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

getLandmarks

```
public Enumeration getLandmarks()
  throws IOException
```

Lists all landmarks stored in the store.

Returns:

an `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

getLandmarks

```
public Enumeration getLandmarks(String category,
         double minLatitude,
         double maxLatitude,
         double minLongitude,
         double maxLongitude)
  throws 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`

getLandmark

```
public Landmark getLandmark(int identifier)
  throws IOException
```

Returns the `Landmark` object for the given identifier, `null` if the `LandmarkStore` does not contain matching landmark.

(continued from last page)

Parameters:
`identifier` - the landmark's identifier

Returns:
the requested `Landmark` or `null` if the landmark can not be found

Throws:
`IllegalArgumentException` - if `identifier 0`
`IOException` - if an I/O error happened when accessing the landmark store

getNumberOfLandmarks

```
public int getNumberOfLandmarks(String category)
  throws IOException
```

This convenience method returns the number of landmarks in the given category. If the category is `null` the method returns the number of landmarks in the landmark store. Applications may use this method, for example, to check if the number of landmarks has changed after receiving `LandmarkStoreListener.landmarkStoreUpdated` notification.

Parameters:
`category` - the landmark category, from which the landmark number is requested, `null` if the total number of landmarks in the landmark store is requested

Returns:
the number of landmarks in the given category or in the landmark store

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

removeLandmarkFromCategory

```
public void removeLandmarkFromCategory(Landmark lm,
         String category)
  throws 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`

(continued from last page)

removeLandmarkFromAllCategories

```
public void removeLandmarkFromAllCategories(Landmark landmark)
  throws IOException
```

Removes all the categories associated to the specified landmark.

Parameters:

`landmark` - landmark whose categories are removed

Throws:

`NullPointerException` - if `landmark` is `null`

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

updateLandmark

```
public void updateLandmark(Landmark landmark)
  throws IOException,
         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.

If there is a `LandmarkStoreListener` assigned for this landmark store, the API implementation **must** send `LandmarkStoreListener.landmarkStoreUpdated` notification to registered applications before this method returns.

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

deleteLandmark

```
public void deleteLandmark(Landmark landmark)
  throws IOException,
         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:

(continued from last page)

`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`

getCategories

```
public 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 `Enumeration` containing `Strings` representing the category names. If there are no categories defined in this `LandmarkStore`, an `Enumeration` with no entries is returned.

addCategory

```
public void addCategory(String categoryName)
  throws LandmarkException,
         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

deleteCategory

```
public void deleteCategory(String categoryName)
  throws LandmarkException,
         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

(continued from last page)

`SecurityException` - if the application does not have the permission to manage categories

getCategoriesOfLandmark

```
public Enumeration getCategoriesOfLandmark(Landmark landmark)
  throws IOException
```

Returns the landmark categories for the specified landmark. If the specified landmark does not belong to any landmark category, an empty `Enumeration` is returned.

Parameters:

`landmark` - landmark whose categories are queried

Returns:

an `Enumeration` of `String` objects that specify categories of the landmark, an empty `Enumeration` if the landmark does not belong to any category

Throws:

`NullPointerException` - if `landmark` is `null`

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

addLandmarkStoreListener

```
public void addLandmarkStoreListener(LandmarkStoreListener listener)
  throws LandmarkException
```

Adds a `LandmarkStoreListener` that is notified when there are changes in the specified landmark store. The listener is removed with method `removeLandmarkStoreListener` method. There **may** be more than one `LandmarkStoreListener` set at a time. If an application tries to add a listener that is already set, this method returns without actions.

If this method is called with a listener that is `null`, the method returns silently without any actions.

Parameters:

`listener` - a listener for the changes in the landmark store

Throws:

`LandmarkException` - if the landmark store implementation does not support adding listeners to the landmark store

removeLandmarkStoreListener

```
public void removeLandmarkStoreListener(LandmarkStoreListener listener)
```

Removes the specified listener from receiving notifications about changes in the landmark store.

If this method is called with a listener that is `null`, or with a listener that has not been set this method return silently without any actions. If the landmark store implementation does not support landmark store listeners, the method returns silently without any actions.

Parameters:

`listener` - listener to be removed from receiving notifications

com.sirf.microedition.location Interface LandmarkStoreListener

public interface **LandmarkStoreListener**
extends

This interface represents a listener that receives events associated with changes in the landmark store. Applications implement this interface and register it with `LandmarkStore.addLandmarkStoreListener` method to obtain notifications about changes in the landmark store.

Method Summary

`void`	`landmarkStoreUpdated(String storeName)` Called by the platform when there is a change of any kind (addition, removal, update of a landmark) in the specified landmark store.

Methods

landmarkStoreUpdated

`public void **landmarkStoreUpdated**(String storeName)`

Called by the platform when there is a change of any kind (addition, removal, update of a landmark) in the specified landmark store.

Parameters:
`storeName` - the landmark store that has been changed

com.sirf.microedition.location
Class Location

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Location
```

public class **Location**
extends Object

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.

Field Summary

`protected`	`addressInfo`
`protected`	`coordinates`
`protected`	`course`
`protected`	`extraInfo`
`protected`	`isValid`
`protected`	`locationMethod`
`public static final`	`MTA_ASSISTED` Location method is assisted by the other party (Terminal assisted for Network based, Network assisted for terminal based). Value: **`262144`**
`public static final`	`MTA_UNASSISTED` Location method is unassisted. Value: **`524288`**

public static final	MTE_ANGLEOFARRIVAL Location method Angle of Arrival for cellular / terrestrial RF system. Value: **32**
public static final	MTE_CELLID Location method Cell-ID for cellular (in GSM, this is the same as CGI, Cell Global Identity). Value: **8**
public static final	MTE_SATELLITE Location method using satellites (for example, Global Positioning System (GPS)). Value: **1**
public static final	MTE_SHORTRANGE Location method Short-range positioning system (for example, Bluetooth LP). Value: **16**
public static final	MTE_TIMEDIFFERENCE Location method Time Difference for cellular / terrestrial RF system (for example, Enhanced Observed Time Difference (E-OTD) for GSM). Value: **2**
public static final	MTE_TIMEOFARRIVAL Location method Time of Arrival (TOA) for cellular / terrestrial RF system. Value: **4**
public static final	MTY_NETWORKBASED Location method is of type network based. Value: **131072**
public static final	MTY_TERMINALBASED Location method is of type terminal based. Value: **65536**
protected	speed
protected	timestamp

Constructor Summary

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

Method Summary

AddressInfo	getAddressInfo() Returns the `AddressInfo` associated with this `Location` object.
float	getCourse() Returns the terminal's course made good in degrees relative to true north.
String	getExtraInfo(String mimetype) Returns extra information about the location.

int	getLocationMethod() Returns information about the location method used.
QualifiedCoordinates	getQualifiedCoordinates() Returns the coordinates of this location and their accuracy.
float	getSpeed() Returns the terminal's current ground speed in meters per second (m/s) at the time of measurement.
long	getTimestamp() Returns the timestamp at which the data was collected.
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.

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

Fields

MTE_SATELLITE

`public static final int` **`MTE_SATELLITE`**

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

`MTE_SATELLITE = 0x00000001`
Constant value: **1**

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`
Constant value: **2**

MTE_TIMEOFARRIVAL

`public static final int` **`MTE_TIMEOFARRIVAL`**

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

`MTE_TIMEOFARRIVAL = 0x00000004`
Constant value: **4**

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`
Constant value: **`8`**

MTE_SHORTRANGE

`public static final int` **`MTE_SHORTRANGE`**

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

`MTE_SHORTRANGE = 0x00000010`
Constant value: **`16`**

MTE_ANGLEOFARRIVAL

`public static final int` **`MTE_ANGLEOFARRIVAL`**

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

`MTE_ANGLEOFARRIVAL = 0x00000020`
Constant value: **`32`**

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`
Constant value: **`65536`**

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`
Constant value: **`131072`**

MTA_ASSISTED

`public static final int` **`MTA_ASSISTED`**

(continued from last page)

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

`MTA_ASSISTED = 0x00040000`
Constant 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`
Constant value: **`524288`**

isValid

```
protected boolean isValid
```

timestamp

```
protected long timestamp
```

coordinates

```
protected com.sirf.microedition.location.QualifiedCoordinates coordinates
```

speed

```
protected float speed
```

course

```
protected float course
```

locationMethod

```
protected int locationMethod
```

addressInfo

```
protected com.sirf.microedition.location.AddressInfo addressInfo
```

extraInfo

`protected java.lang.String **extraInfo**`

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

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

See Also:
`getExtraInfo(String)`

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

getQualifiedCoordinates

`public QualifiedCoordinates **getQualifiedCoordinates**()`

(continued from last page)

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

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

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

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

getExtraInfo

`public String **getExtraInfo**(String mimetype)`

(continued from last page)

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.

com.sirf.microedition.location
Class LocationException

```
java.lang.Object
   |
   +-java.lang.Throwable
         |
         +-java.lang.Exception
               |
               +-com.sirf.microedition.location.LocationException
```

All Implemented Interfaces:
Serializable

public class **LocationException**
extends Exception

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.

Constructor Summary

`public`	`LocationException()` Constructs a `LocationException` with no detail message.
`public`	`LocationException(String s)` Constructs a `LocationException` with the specified detail message.

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, toString, wait, wait, wait`

Constructors

LocationException

`public` **`LocationException`**`()`

Constructs a `LocationException` with no detail message.

LocationException

`public` **`LocationException`**`(String s)`

Constructs a `LocationException` with the specified detail message.

Parameters:
`s` - the detail message

com.sirf.microedition.location
Interface LocationListener

public interface **LocationListener**
extends

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.

Method Summary

`void`	`locationUpdated(LocationProvider provider, Location location)` Called by the `LocationProvider` to which this listener is registered.
`void`	`providerStateChanged(LocationProvider provider, int newState)` Called by the `LocationProvider` to which this listener is registered if the state of the `LocationProvider` has changed.

Methods

locationUpdated

```
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

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

(continued from last page)

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.

com.sirf.microedition.location
Class LocationProvider

```
java.lang.Object
  |
  +-com.sirf.microedition.location.LocationProvider
```

public abstract class **LocationProvider**
extends Object

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 Locations.

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.

Field Summary

`public static final`	`AVAILABLE` Availability status code: the location provider is available. Value: **1**
`public static final`	`OUT_OF_SERVICE` Availability status code: the location provider is out of service. Value: **3**
`public static final`	`TEMPORARILY_UNAVAILABLE` Availability status code: the location provider is temporarily unavailable. Value: **2**

Constructor Summary

`protected`	`LocationProvider()` Empty constructor to help implementations and extensions.

Method Summary

`static void`	`addProximityListener(ProximityListener listener, Coordinates coordinates, float proximityRadius)` Adds a ProximityListener for updates when proximity to the specified coordinates is detected.
`static LocationProvider`	`getInstance(Criteria criteria)` This factory method is used to get an actual LocationProvider implementation based on the defined criteria.
`static Location`	`getLastKnownLocation()` Returns the last known location that the implementation has.
`abstract Location`	`getLocation(int timeout)` Retrieves a Location with the constraints given by the Criteria associated with this class.

`abstract int`	`getState()` Returns the current state of this LocationProvider.
`static void`	`removeProximityListener(ProximityListener listener)` Removes a ProximityListener from the list of recipients for updates.
`abstract void`	`reset()` Resets the LocationProvider.
`abstract void`	`setLocationListener(LocationListener listener, int interval, int timeout, int maxAge)` Adds a LocationListener for updates at the defined interval.

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

Fields

AVAILABLE

`public static final int` **`AVAILABLE`**

Availability status code: the location provider is available.
Constant value: **`1`**

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.
Constant value: **`2`**

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.
Constant value: **`3`**

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

(continued from last page)

getState

```
public abstract 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

getInstance

```
public static LocationProvider getInstance(Criteria criteria)
  throws 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't 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 isn't 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 LocationProviders are currently out of service

SiRF Implementation Note

LocationException - if Configuration details for Network Server and Serial port has not been configured before calling this function.

See Also:
`Criteria`

(continued from last page)

getLocation

```
public abstract Location getLocation(int timeout)
  throws LocationException,
         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't 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:

`LocationException` - if the location couldn't be retrieved or if the timeout period expired

`java.lang.InterruptedException` - if the operation is interrupted by calling reset() from another thread

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

`java.lang.IllegalArgumentException` - if the timeout = 0 or timeout < -1

setLocationListener

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

(continued from last page)

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 Locations at the defined interval until the first location result is available.

The timeout parameter determines a timeout that is used if it's 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's 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.

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 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 the interval is 0, this parameter has no effect.

Throws:

`java.lang.IllegalArgumentException` - if interval < -1, or if (interval != -1) and (timeout > interval or maxAge > interval or (timeout < 1 and timeout != -1) or (maxAge < 1 and maxAge != -1))
`java.lang.SecurityException` - if the calling application does not have a permission to query the location information

reset

`public abstract void **reset**()`

Resets the LocationProvider.

All pending synchronous location requests will be aborted and any blocked 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.

(continued from last page)

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 doesn't have any previous location information.

Throws:

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

addProximityListener

```
public static void addProximityListener(ProximityListener listener,
          Coordinates coordinates,
          float proximityRadius)
  throws 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't be currently done due to the current state of the method used to implement it, this method shall succeeed and the 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:

`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

`java.lang.IllegalArgumentException` - if the proximity radius is 0 or negative* or Float.NaN

`java.lang.NullPointerException` - if the listener or coordinates parameter is null

`java.lang.SecurityException` - if the application does not have the permission to register a proximity listener

(continued from last page)

removeProximityListener

```
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

com.sirf.microedition.location
Class Orientation

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Orientation
```

public class **Orientation**
extends Object

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).

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. Providing the pitch and roll is optional. Most commonly, this class will be used to obtain the current compass direction.

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.

Constructor Summary

`public`	`Orientation(float azimuth, boolean isMagnetic, float pitch, float roll)` Constructs a new Orientation object with the compass azimuth, pitch and roll parameters specified.

Method Summary

`float`	`getCompassAzimuth()` Returns the terminal's horizontal compass azimuth in degrees relative to either magnetic or true north.
`static Orientation`	`getOrientation()` Returns the terminal's current orientation.
`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.
`float`	`getRoll()` Returns the terminal's rotation in degrees around its own longitudinal axis.
`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.

Methods inherited from class `java.lang.Object`

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

Constructors

Orientation

```
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` - 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

See Also:

`isOrientationMagnetic()`

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 is relative 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

(continued from last page)

See Also:
`getCompassAzimuth()`

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

getOrientation

```
public static Orientation getOrientation()
  throws LocationException
```

Returns the terminal's current orientation.

Returns:
returns an Orientation object containing the terminal's current orientation or null if the orientation can't be currently determined

Throws:
`LocationException` - if the implementation does not support orientation determination
`java.lang.SecurityException` - if the calling application does not have a permission to query the orientation

See Also:
`Orientation`

com.sirf.microedition.location
Interface ProximityListener

public interface **ProximityListener**
extends

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 notfications 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 may be called several times to notify the application about the state change in the monitoring by sending `monitoringStateChanged` notification. The listener is called only once when the terminal enters the proximity of the registered coordinates. The registration with these coordinates is cancelled 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.

Method Summary

`void`	`monitoringStateChanged(boolean isMonitoringActive)` Called to notify that the state of the proximity monitoring has changed.
`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.

Methods

proximityEvent

```
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

monitoringStateChanged

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

(continued from last page)

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.

com.sirf.microedition.location
Class QualifiedCoordinates

```
java.lang.Object
  |
  +-com.sirf.microedition.location.Coordinates
      |
      +-com.sirf.microedition.location.QualifiedCoordinates
```

public class **QualifiedCoordinates**
extends Coordinates

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

Fields inherited from class `com.sirf.microedition.location.Coordinates`

DD_MM, DD_MM_SS

Constructor Summary

`public`	`QualifiedCoordinates(double latitude, double longitude, float altitude, float horizontalAccuracy, float verticalAccuracy)` Constructs a new QualifiedCoordinates object with the values specified.

Method Summary

`float`	`getHorizontalAccuracy()` Returns the horizontal accuracy of the location in meters (1-sigma standard deviation).
`float`	`getVerticalAccuracy()` Returns the accuracy of the location in meters in vertical direction (orthogonal to ellipsoid surface, 1-sigma standard deviation).
`void`	`setHorizontalAccuracy(float horizontalAccuracy)` Sets the horizontal accuracy of the location in meters (1-sigma standard deviation).
`void`	`setVerticalAccuracy(float verticalAccuracy)` Sets the accuracy of the location in meters in vertical direction (orthogonal to ellipsoid surface, 1-sigma standard deviation).

Methods inherited from class `com.sirf.microedition.location.Coordinates`

`azimuthTo, convert, convert, distance, getAltitude, getLatitude, getLongitude, setAltitude, setLatitude, setLongitude, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

(continued from last page)

QualifiedCoordinates

```
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:

`java.lang.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.

(continued from last page)

setHorizontalAccuracy

`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:

`java.lang.IllegalArgumentException` - if the parameter is less than 0

setVerticalAccuracy

`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:

`java.lang.IllegalArgumentException` - if the parameter is less than 0

Package
com.sirf.microedition.location.extensions

com.sirf.microedition.location.extensions
Class BTGPSDiscoverer

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.BTGPSDiscoverer
```

public class **BTGPSDiscoverer**
extends Object

This class provides method to search for Bluetooth GPS device.

This can be used to search Bluetooth enabled GPS devices and gives the address details of the remote GPS Device which is used for serial connection over Bluetooth to connect to the BT-GPS device.

Constructor Summary

`public`	`BTGPSDiscoverer(BTGPSDiscovererListener btListener)` Creates a new instance of `BTGPSDiscoverer`.

Method Summary

`void`	`close()` This method cancels any outstanding service search or inquiry on the BT-GPS device.
`void`	`searchBTGPSDevices()` This method searches the Bluetooth GPS devices.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

BTGPSDiscoverer

`public` **`BTGPSDiscoverer`**`(BTGPSDiscovererListener btListener)`

Creates a new instance of `BTGPSDiscoverer`.

Parameters:
`btListener` - the listener to be registered.

Throws:
`java.lang.NullPointerException` - if the listener parameter is null.

Methods

(continued from last page)

searchBTGPSDevices

```
public void searchBTGPSDevices()
```

This method searches the Bluetooth GPS devices. Limitation: Current implementation supports search of a BT-GPS device whose name starts with "BT-GPS". After having located one such device the search stops giving the address of the discovered BT-GPS device to the application.

close

```
public void close()
```

This method cancels any outstanding service search or inquiry on the BT-GPS device.

com.sirf.microedition.location.extensions
Interface BTGPSDiscovererListener

public interface **BTGPSDiscovererListener**
extends

This interface collects methods that enable the application to receive asynchronous events generated by `BTGPSDiscoverer` to notify Bluetooth GPS device discovery status.
Application may implement the interface and register their implementation with the `BTGPSDiscoverer` constructor.

Method Summary

`void`	`notifyDevicesDiscovered(String deviceConnectionStrings, String deviceFriendlyNames)` A callback method that will be called when the `BTGPSDiscoverer` has completed the device discovery, and has found at least one BT-GPS device that supports SerialPort Profile.
`void`	`notifyDiscoveryError(int errorCode, String errorMessage)` A callback to notify that an error has occurred while trying to discover BT devices.

Methods

notifyDevicesDiscovered

```
public void notifyDevicesDiscovered(String deviceConnectionStrings,
        String deviceFriendlyNames)
```

A callback method that will be called when the `BTGPSDiscoverer` has completed the device discovery, and has found at least one BT-GPS device that supports SerialPort Profile.

Parameters:
`deviceConnectionStrings` - A connection string that can be used to connect to the found device. Currently this string is hard-wired to start with "BT-GPS".
`deviceFriendlyNames` - Containing human readable names for the found devices, if the found device has defined one. These names can be shown to the user.

notifyDiscoveryError

```
public void notifyDiscoveryError(int errorCode,
        String errorMessage)
```

A callback to notify that an error has occurred while trying to discover BT devices.

Parameters:
`errorCode` - A code for the error.
`errorMessage` - An error message .

com.sirf.microedition.location.extensions
Class LocationProviderConfiguration

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.LocationProviderConfiguration
```

public final class **LocationProviderConfiguration**
extends Object

The `Configuration` class provides APIs to configure AGPS Server, its port and the GPS serial communication or set the BT-GPS device address.

Configuration settings are mandatory.

Method Summary

`static void`	`setAgpsServer(String ipAddress, int portNumber)` This function is used to configure the AGPS server IP address and port number.
`static void`	`setApproxPosition(boolean approxPostionAvailable, double approxLatitude, double approxLongitude)`
`static void`	`setSerialConnectionString(String serialConfig)` This function is used to configure the serial communication to the GPS device.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

setAgpsServer

```
public static void setAgpsServer(String ipAddress,
         int portNumber)
```

This function is used to configure the AGPS server IP address and port number.

Parameters:
`ipAddress` - IP address of the AGPS server
`portNumber` - port number of the AGPS server. Valid range:[1024, 65535]

Throws:
`java.lang.IllegalArgumentException` - if the ipAddress or portNumber is invalid
`java.lang.NullPointerException` - if the ipAddress parameter is null

(continued from last page)

setSerialConnectionString

```
public static void setSerialConnectionString(String serialConfig)
```

This function is used to configure the serial communication to the GPS device.
It can be used to specify either a standard serial port or a serial port profile over Bluetooth.

Parameters:

`serialConfig` - the serial configuration string If a standard serial port is used, this parameter must contain only the name of the port (eg. "COM1" or "1"). The exact format and value of the device name is platform dependent.
If a Bluetooth device is to be used, then this parameter must contain the complete connection string used to create a Bluetooth connection to the desired device, including the protocol name (eg. "btspp://...." Say the BT address is 00:0D:B5:31:BD:D8 where the NAP part of the address is: 00:0D, UAP is B5:31 and LAP is BD:B8 The connection string is then btspp://000DB531BDB8:1;authenticate=false;encrypt=false;master=false). This connection string is normally obtained via the Bluetooth API on the device. It is the applications' responsibility to obtain the correct string.

Throws:

`java.java.lang.NullPointerException` - if the serialConfig parameter is null

setApproxPosition

```
public static void setApproxPosition(boolean approxPostionAvailable,
         double approxLatitude,
         double approxLongitude)
```

com.sirf.microedition.location.extensions
Interface NMEALocationListener

public interface **NMEALocationListener**
extends

`Like LocationListener, NMEAListener` represents a listener that receives events associated with a particular NMEAProvider. Application may implement this interface and register it with a NMEAProvider to obtain regular NMEA updates.

The implementation provides NMEA (National Marine Electronic Association) updates as and when it receives NMEA information from the GPS receiver. NMEA updates is not possible at the time when LocationProvider state is OUT_OF_SERVICE.

Method Summary

`void`	`updateNMEA(String sentence)` This method is called by the NMEAProvider, to which this listener is registered.

Methods

updateNMEA

`public void **updateNMEA**(String sentence)`

This method is called by the NMEAProvider, to which this listener is registered.

Parameters:
`sentence` - a String containing the NMEA sentence .

com.sirf.microedition.location.extensions
Class NMEALocationProvider

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.NMEALocationProvider
```

public final class **NMEALocationProvider**
extends Object

The `NMEAProvider` implements the NMEA-providing module, generating NMEA sentences for registered listeners.

An application can register NMEAListeners to a NMEAProvider for receiving NMEA sentences.

This class provides static methods to register and deregister NMEAListener using addNMEAListener and removeNMEAListener methods respectively.

As such there is no limit to register a number of listeners but it is advisable not to register more than five listener at any given time - limitation may be posed by underlined platform.

Method Summary

`static void`	`addNMEAListener(NMEALocationListener listener)` This method adds an NMEAListener to get continuous NMEA message stream from the NMEAProvider.
`static void`	`removeNMEAListener(NMEALocationListener listener)` Removes an NMEAListener from the list of recipients for NMEA updates.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

addNMEAListener

```
public static void addNMEAListener(NMEALocationListener listener)
  throws LocationException
```

This method adds an NMEAListener to get continuous NMEA message stream from the NMEAProvider.

Parameters:
`listener` - the listener to be registered

Throws:
`LocationException` - if the platform does not have resources to add a new listener
`java.lang.NullPointerException` - if the listener parameter is null
`LocationException` - if the GPS configuration has not been set.

(continued from last page)

removeNMEAListener

```
public static void removeNMEAListener(NMEALocationListener listener)
```

Removes an NMEAListener from the list of recipients for NMEA 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 be removed.

com.sirf.microedition.location.extensions Interface POIServiceProvider

All Superinterfaces:
ServiceProvider

public interface **POIServiceProvider**
extends ServiceProvider

Interface for service providers that provide directory search services. The actual database that is used in searching is determined at the SiRFStudio server configuration.

If the used provider supports categories, all valid category names can be retrieved using `POIServiceProvider.getCategories()` method.

Method Summary

`String[]`	`getCategories()` Returns the category names that are defined in this `POIServiceProvider`.
`Landmark[]`	`getPOIs(String category, String name, GeographicArea area, int maxResponses)` Gets the POIs over the network from configured database.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceProvider`
`configureProvider, getLanguage, getProviderInfo, getTimeout, reset, setLanguage, setTimeout, supportsArea`

Methods

getPOIs

```
public Landmark[] getPOIs(String category,
                          String name,
                          GeographicArea area,
                          int maxResponses)
   throws ServiceException,
          InterruptedException
```

Gets the POIs over the network from configured database. The results include POIs where the category and/or name matches the given parameters.

This is a blocking method, and it will return after the results are received from the network, or if error or timeout occurs.

Parameters:

`category` - the category of the POI, `null` implies a wildcard that matches all categories. The category passed must be one returned from the `getCategories()` method.
`name` - the name of the desired POI, `null` implies a wildcard that matches all the names within the category indicated by the category parameter.
`area` - Geographic area that is used as the filter. All returned POIs must lie within this area. May be null.
`maxResponses` - the maximum number of responses the application expects to be returned.

(continued from last page)

Returns:

an `Landmark[]` containing all the matching POIs as landmarks or empty`Landmark[]` if no POI matched the given parameters

Throws:

`ServiceException` - if this ServiceProvider cannot service the request at this point.
`IllegalArgumentException` - if maxResponses < 1
`InterruptedException` - if the request has been canceled by the application

getCategories

```
public String[] getCategories()
  throws ServiceException
```

Returns the category names that are defined in this `POIServiceProvider`. The names shall be such that they can be displayed to the end user and have a meaning to the end user. Note that this method may block for a while, since categories may be retrieved from a server.

Returns:

An `String[]` containing `Strings` representing the category names. If there are no categories defined in this `POIServiceProvider`, an empty `String[]` is returned.

Throws:

`ServiceException` - if there is an error retrieving the categories.

com.sirf.microedition.location.extensions
Class SatelliteContext

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.SatelliteContext
```

public class **SatelliteContext**
extends Object

The `SatelliteContext` class contains the additional information for Satellites that are currently being acquired or tracked, or have been used in the location computation.

Copyright (c): SiRF Technology

Constructor Summary

`protected`	`SatelliteContext()` Creates a new instance of SatelliteContext

Method Summary

`byte`	`getNumSatsUsed()` Returns the number of Satellites contained in the context.
`SatelliteInfo`	`getSatelliteInfo(byte index)` Returns a SatelliteInfo object containing specific information about one of the Satellites being acquired or tracked, or being used in the Location computation.

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

Constructors

SatelliteContext

`protected` **`SatelliteContext()`**

Creates a new instance of SatelliteContext

Methods

getNumSatsUsed

`public byte` **`getNumSatsUsed()`**

Returns the number of Satellites contained in the context.

Returns:
Value of property numSatsUsed. This is an integer number with range from 0 to 12.

getSatelliteInfo

```
public SatelliteInfo getSatelliteInfo(byte index)
  throws LocationException
```

Returns a SatelliteInfo object containing specific information about one of the Satellites being acquired or tracked, or being used in the Location computation.

Parameters:

`index` - index of the Satellite for which to get information. Must be between 0 and the number of Satellites reported by getNumSatsUsed().

Returns:

An instance of SatelliteInfo containing information about one Satellite.

com.sirf.microedition.location.extensions
Class SatelliteInfo

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.SatelliteInfo
```

public class **SatelliteInfo**
extends Object

The `SatelliteInfo` class is used by the SatelliteContext to provide information related to one Satellite used in computing the associated Location.

Constructor Summary

`public`	`SatelliteInfo()` Creates a new instance of SatelliteInfo

Method Summary

`short`	`getAzimuth()` Returns the Azimuth for this Satellite.
`short`	`getElevationAngle()` Returns the Elevation Angle for this Satellite.
`byte`	`getSatelliteId()` Returns the ID for this Satellite.
`byte`	`getSignalStrength()` Returns the Signal Strength for this Satellite.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

SatelliteInfo

`public SatelliteInfo()`

Creates a new instance of SatelliteInfo

Methods

getSatelliteId

`public byte getSatelliteId()`

Returns the ID for this Satellite.

(continued from last page)

Returns:
Value of property satId. This is an integer number indicating the satellite id.

getSignalStrength

`public byte getSignalStrength()`

Returns the Signal Strength for this Satellite.

Returns:
Value of property satCNo. This is an integer number indicating the satellite's signal-to-noise ratio in dBHz. The range is from 0 to 90.

getAzimuth

`public short getAzimuth()`

Returns the Azimuth for this Satellite.

Returns:
Value of property azimuth. This is an integer number indicating the azimuth of the associated satellite in degress. The range is from 0 to 359.

getElevationAngle

`public short getElevationAngle()`

Returns the Elevation Angle for this Satellite.

Returns:
Value of property elAngle. This is an integer number indicating the elevation angle of the associated satellite in degress. The range is from 0 to 90.

com.sirf.microedition.location.extensions
Interface SatelliteInfoListener

public interface **SatelliteInfoListener**
extends

Application may implement this interface and register it with a SatelliteInfoProvider to get the information about the Satellites which being acquired or tracked, or being used in the Location computation.

Satellites updates is not possible when LocationProvider state is OUT_OF_SERVICE.

Method Summary

`void`	`updateSatelliteInfo(SatelliteContext context)`

Methods

updateSatelliteInfo

```
public void updateSatelliteInfo(SatelliteContext context)
```

com.sirf.microedition.location.extensions
Class SatelliteInfoProvider

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.SatelliteInfoProvider
```

public class **SatelliteInfoProvider**
extends Object

The `SatelliteInfoProvider` provide method to register and remove SateliiteInfoListener.

An application can register SatelliteInfoListener for receiving an information about the Satellites which are used in Location computation.

SatelliteInfoProvider limits application to register maximum of five satelliteinfoListener at any given time.

Method Summary

`void`	`addSatlliteInfoListener(SatelliteInfoListener listener)` This method adds an SatelliteInfoListener to get Satellites information.if the resgietred listener count is reached to maximum allowed then this method silently return with no action.
`static SatelliteInfoProvider`	`getInstance()` Static Factory method to gets the singleton instance of the SatelliteInfoProvider Class.
`byte`	`getNumSatsUsed()` Returns the number of Satellites contained in the context.
`void`	`removeSatlliteInfoListener(SatelliteInfoListener listener)` Removes an SatelliteInfoListener from the list of registered listener.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getInstance

`public static SatelliteInfoProvider getInstance()`

Static Factory method to gets the singleton instance of the SatelliteInfoProvider Class.

Returns:
instance Singleton Object of SatelliteInfoProvider class.

getNumSatsUsed

`public byte getNumSatsUsed()`

Returns the number of Satellites contained in the context.

(continued from last page)

Returns:
Value of property numSatsUsed. This is an integer number with range from 0 to 12.

addSatlliteInfoListener

```
public void addSatlliteInfoListener(SatelliteInfoListener listener)
  throws LocationException
```

This method adds an SatelliteInfoListener to get Satellites information.if the resgietred listener count is reached to maximum allowed then this method silently return with no action.

Parameters:
`listener` - the listener to be registered

Throws:
`LocationException` - if the platform does not have resources to add a new listener.
`java.lang.NullPointerException` - if the listener parameter is null
`LocationException` - if the GPS configuration has not been set.

removeSatlliteInfoListener

```
public void removeSatlliteInfoListener(SatelliteInfoListener listener)
```

Removes an SatelliteInfoListener from the list of registered listener. 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 be removed.

com.sirf.microedition.location.extensions
Class SiRFProviderManager

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.SiRFProviderManager
```

public abstract class **SiRFProviderManager**
extends Object

This class is similar to `ProviderManager` so that applications can get instances of SiRF related ServiceProviders.

If the application need to get an instance of eg. `POIServiceProvider`, it has to get it using this class's methods instead of `ProviderManager` methods.

Field Summary

`public static final`	`COM_SIRF_STUDIO_POI_CATEGORIES` Value: **`com.sirf.studio.poi.categories`**
`public static final`	`COM_SIRF_STUDIO_POI_CATEGORY_ID` Value: **`com.sirf.studio.poi.category.id`**
`public static final`	`POI` Service provider type for POI services Value: **`8`**

Constructor Summary

`public`	`SiRFProviderManager()`

Method Summary

`static ServiceProvider`	`connectToServiceProvider(ProviderInfo provider, String extraInfo)` Method to connect to a service provider.
`static ServiceProvider`	`connectToServiceProvider(String name, int type, String extraInfo)` Method to connect to a service provider.
`static ProviderInfo[]`	`findServiceProviders(int type, String extraInfo)` Method to get all ProviderInfo objects for service provider type POI .

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

(continued from last page)

COM_SIRF_STUDIO_POI_CATEGORIES

`public static final java.lang.String` **`COM_SIRF_STUDIO_POI_CATEGORIES`**

Constant value: **`com.sirf.studio.poi.categories`**

COM_SIRF_STUDIO_POI_CATEGORY_ID

`public static final java.lang.String` **`COM_SIRF_STUDIO_POI_CATEGORY_ID`**

Constant value: **`com.sirf.studio.poi.category.id`**

POI

`public static final int` **`POI`**

Service provider type for POI services
Constant value: **`8`**

Constructors

SiRFProviderManager

`public` **`SiRFProviderManager`**`()`

Methods

findServiceProviders

```
public static ProviderInfo[] findServiceProviders(int type,
          String extraInfo)
  throws ServiceException
```

Method to get all ProviderInfo objects for service provider type POI . This method is similar to `ProviderManager.findServiceProviders(int, String)` method, but it only allows retrieving ProviderInfo's for services of type POI.

Parameters:

`type` - Type of service of which ProviderInfo's are requested, must be type POI defined in this class.

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

an array of `ProviderInfo` objects for all POI service providers, an empty `ProviderInfo[]` if there are no providers available or configured for this application, or an error occured when retrieving the ProviderInfos.

Throws:

`ServiceException` - if service request fails, for example, if information about services can not be fetched from remote server
`IllegalArgumentException` - if type passed in is not type POI defined in this class.

(continued from last page)

connectToServiceProvider

```
public static ServiceProvider connectToServiceProvider(String name,
         int type,
         String extraInfo)
```

Method to connect to a service provider. This method is similar to the one defined in `ProviderManager.connectToServiceProvider(String, int, String)` but it allows only connect to service providers of type POI defined in this class. See `ProviderManager.connectToServiceProvider(String, int, String)` for more info.

Parameters:

`name` - of the provider to connect to.

`type` - Type of service provider. Must be `SiRFProviderManager.POI` .

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

the instance of the requested service provider, `null` if there is no providers with given name.

Throws:

`java.lang.IllegalArgumentException` - if `type` is not constant POI

connectToServiceProvider

```
public static ServiceProvider connectToServiceProvider(ProviderInfo provider,
         String extraInfo)
```

Method to connect to a service provider. This method is similar to the one defined in `ProviderManager.connectToServiceProvider(ProviderInfo, String)` but it allows only connect to service providers of type POI defined in this class. See `ProviderManager.connectToServiceProvider(ProviderInfo, String)` for more info.

Parameters:

`provider` - the service provider to be instantiated

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

the instance of the requested service provider, `null` if the API implementation does not support requested service type

Throws:

`java.lang.NullPointerException` - if `provider` is `null`

com.sirf.microedition.location.extensions
Class SystemConfiguration

```
java.lang.Object
  |
  +-com.sirf.microedition.location.extensions.SystemConfiguration
```

public class **SystemConfiguration**
extends Object

This class is a central point to configure system parameters.

Field Summary

public static	`backgroungImage` The image that is rendered in the background on map screens.
public static	`display` Handle to Display object.
public static	`emulateNetworkFailure` Flag for network testing mode.
public static	`highlightedLandmarkIcon` The image that is rendered to present a landmark that is highlighted.
public static	`isDebugMode`
public static	`landmarkIcon` The image that is rendered to present a landmark that is not selected/highlighted.
public static	`landmarkIconsAnchor` The anchor point for all landmark icons.
public static	`locationIcon` The image that is rendered to present a location (coordinates).
public static	`mediumFont` The font used as the medium font throughout the entire system.
public static	`routeEndIcon` The image that is rendered to present the route end point.
public static	`routeIconsAnchor` The anchor point for all route icons.
public static	`routeStartIcon` The image that is rendered to present the route start point.
public static	`routeWaypointIcon` The image that is rendered to present the route waypoint.
public static	`selectedLandmarkIcon` The image that is rendered to present a landmark that is selected.

`public static`	`smallFont` The font used as the small font throughout the entire system.

Constructor Summary

`public`	`SystemConfiguration()`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

display

`public static javax.microedition.lcdui.Display` **`display`**

Handle to Display object.

This reference enables the service providers that require access to UI to get a handle to `Display`, so that they can take over the UI. This must be set by the MIDlet in the very beginning in order to give API access to application's Display.

This needs to be set only when this library is bundled with developers application (opposed to deployment scenario where this library is part of the pre-installed class libraries).

smallFont

`public static javax.microedition.lcdui.Font` **`smallFont`**

The font used as the small font throughout the entire system.

mediumFont

`public static javax.microedition.lcdui.Font` **`mediumFont`**

The font used as the medium font throughout the entire system.

isDebugMode

`public static boolean` **`isDebugMode`**

backgroungImage

`public static javax.microedition.lcdui.Image` **`backgroungImage`**

The image that is rendered in the background on map screens.

locationIcon

`public static javax.microedition.lcdui.Image` **`locationIcon`**

The image that is rendered to present a location (coordinates).

landmarkIcon

`public static javax.microedition.lcdui.Image` **`landmarkIcon`**

The image that is rendered to present a landmark that is not selected/highlighted.

highlightedLandmarkIcon

`public static javax.microedition.lcdui.Image` **`highlightedLandmarkIcon`**

The image that is rendered to present a landmark that is highlighted. (that is, is on focus)

selectedLandmarkIcon

`public static javax.microedition.lcdui.Image` **`selectedLandmarkIcon`**

The image that is rendered to present a landmark that is selected.

landmarkIconsAnchor

`public static int` **`landmarkIconsAnchor`**

The anchor point for all landmark icons. Must be either Graphics.LEFT|Graphics.BOTTOM or Graphics.HCENTER|Graphics.VCENTER

routeStartIcon

`public static javax.microedition.lcdui.Image` **`routeStartIcon`**

The image that is rendered to present the route start point.

routeWaypointIcon

`public static javax.microedition.lcdui.Image` **`routeWaypointIcon`**

The image that is rendered to present the route waypoint.

routeEndIcon

`public static javax.microedition.lcdui.Image` **`routeEndIcon`**

The image that is rendered to present the route end point.

routeIconsAnchor

`public static int` **`routeIconsAnchor`**

The anchor point for all route icons. Must be either Graphics.LEFT|Graphics.BOTTOM or Graphics.HCENTER|Graphics.VCENTER

emulateNetworkFailure

`public static boolean` **`emulateNetworkFailure`**

Flag for network testing mode. If this is true, the network is effectively disabled (throwing an exception)

Constructors

(continued from last page)

SystemConfiguration

```
public SystemConfiguration()
```

Package com.sirf.microedition.location.services

com.sirf.microedition.location.services
Class CircleGeographicArea

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.CircleGeographicArea
```

All Implemented Interfaces:
GeographicArea

---

public class **CircleGeographicArea**
extends Object
implements GeographicArea

This class represents a circular geographical area in WGS84 coordinate system. Applications can use this object for example to request services from the `MapServiceProvider`.

Fields inherited from interface `com.sirf.microedition.location.services.GeographicArea`
`DOWN`, `LEFT`, `RIGHT`, `UP`

Constructor Summary

`public`	`CircleGeographicArea(Coordinates center, int radius, float direction)` Constructs a `CircleGeographicArea` object.

Method Summary

`void`	`centerOn(Coordinates coords)` Centers this area to given coordinates while preserving the scale and orientation of the area.
`boolean`	`containsCoordinates(Coordinates coordinate)` Checks whether the given coordinate is inside this `CircleGeographicArea` object.
`RectangleGeographicArea`	`getBoundingBox(float orientation)` Returns a `RectangleGeographicArea` object that surrounds this area compeletely.
`Coordinates`	`getCenterPointCoords()` Returns the current coordinates for the center point of this `CircleGeographicArea`.
`float`	`getDirection()` Returns the direction of the geographic area.
`int`	`getRadius()` Returns the current radius of this `CircleGeographicArea`.
`void`	`pan(int panDirection, int newArea)` Pans the geographic area to the given direction.
`void`	`rescale(int percentage)` With this method an application can modify the scale of the geographic area on the map.
`void`	`rotate(float azimuth)` Rotates the geographic area to the given azimuth.

String	toString()

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

Methods inherited from interface `com.sirf.microedition.location.services.GeographicArea`
centerOn, containsCoordinates, getBoundingBox, getDirection, pan, rescale, rotate

Constructors

CircleGeographicArea

```
public CircleGeographicArea(Coordinates center,
                            int radius,
                            float direction)
```

Constructs a `CircleGeographicArea` object. The radius is given in meters. The `direction` specifies what is the orientation of the geographic area. The direction is given as degrees from true north.

Parameters:

`center` - center point of the geographic area
`radius` - radius of the geographic area in meters
`direction` - direction of the geographic area on the map, degrees from true north

Throws:

`java.lang.NullPointerException` - if `center` is `null`
`java.lang.IllegalArgumentException` - if `radius` < 1 or `direction` < 0.0 or `direction` 360.0;
`ServiceException` - if the `GeographicArea` object can not be created

Methods

containsCoordinates

```
public boolean containsCoordinates(Coordinates coordinate)
  throws ServiceException
```

Checks whether the given coordinate is inside this `CircleGeographicArea` object.

Parameters:

`coordinate` - a coordinates to be checked

Returns:

`true`, if the given point is inside this geographical area, else `false`. If the given coordinate is `null`, `false` is retured.

Throws:

`ServiceException` - if the API implementation does not support calculations of containment

getDirection

```
public float getDirection()
```

(continued from last page)

Returns the direction of the geographic area. The value may be the one given as an input parameter in the constructor or value changed with rotate method.

The direction represent the screen up direction on the terminal. For example, when the map image is displayed or created using this object, the service provider **should** generate it that way.

Returns:
the direction of the geographic area

pan

```
public void pan(int panDirection,
            int newArea)
  throws ServiceException
```

Pans the geographic area to the given direction. The `newArea` parameter is the amount of new area in per cents to be included into the geographic area. This parameter is only a request and the API implementation may override it. So if `newArea` is 75, new geographic area contains 75% new area and 25% old area. If the `newArea` > 100%, the new geographic area does not have any old area in it. The `panDirection` **must** be one of the constants defined in this class or a bitwise combination of them. If opposite directions are added to the bitwise combination, those directions are ignored. The pannig is done to the existing `GeographicArea` object instance.

Parameters:
`panDirection` - direction to pan to
`newArea` - the amount of new area after panning in percents

Throws:
`java.lang.IllegalArgumentException` - if `panDirection` is not one of the constants defined in this class or a bitwise combination of them or if `newArea` 1
`ServiceException` - if the `GeographicArea` object can not be panned

rescale

```
public void rescale(int percentage)
  throws ServiceException
```

With this method an application can modify the scale of the geographic area on the map. The new scale is given as percentage of the original size. The rescaling is done to the existing `GeographicArea` object instance.

For example, if the scale of the map is 1:10000 and method `rescale(50)` is called, the resulting scale will be 1:5000 and the map is zoomed in.

Parameters:
`percentage` - percentage of the rescale

Throws:
`java.lang.IllegalArgumentException` - if `percentage` < 1
`ServiceException` - if the `GeographicArea` object can not be rescaled

rotate

```
public void rotate(float azimuth)
  throws ServiceException
```

Rotates the geographic area to the given azimuth. The azimuth is given in degrees relative to true north. Azimuth value is always in the range [0.0, 360.0). The rotating is done to the existing `GeographicArea` object instance.

(continued from last page)

Parameters:
`azimuth` - the rotation azimuth from true north

Throws:
`java.lang.IllegalArgumentException` - if `angle` < 0 or `angle` 360;
`ServiceException` - if the `GeographicArea` object can not be rotated

getCenterPointCoords

```
public Coordinates getCenterPointCoords()
```

Returns the current coordinates for the center point of this `CircleGeographicArea`.

Returns:
the center point coordinates of this circular area.

getRadius

```
public int getRadius()
```

Returns the current radius of this `CircleGeographicArea`.

Returns:
the radius of the circular area

getBoundingBox

```
public RectangleGeographicArea getBoundingBox(float orientation)
  throws ServiceException
```

Returns a `RectangleGeographicArea` object that surrounds this area compeletely.

Parameters:
`direction` - The requested azimuth for the created bounding box.

Returns:
A RectangleGeographicArea object that surrounds this area.

Throws:
`ServiceException` - if RectangleGeographicArea with requested orientation cannot be created.

centerOn

```
public void centerOn(Coordinates coords)
  throws ServiceException
```

Centers this area to given coordinates while preserving the scale and orientation of the area.

Parameters:
`coords` - Coordinates to center on.

Throws:
`ServiceException` - if this area cannot be centered on the passed coordinates.

toString

```
public String toString()
```

(continued from last page)

com.sirf.microedition.location.services
Interface GeocodingServiceProvider

All Superinterfaces:
ServiceProvider

public interface **GeocodingServiceProvider**
extends ServiceProvider

This interface collects the services that geocoding service providers offer. With this interface the application is able to request geocoding and reverse geocoding related services from the service provider. Geocoding means that a service provider converts a street address information into coordinates. In reverse geocoding coordinates are converted into street address information.

Some service providers may only provide either geocoding or reverse geocoding service, but not both. This capability of the geocoding service can be queried from the `ProviderInfo` class with key value `GEO_SP_SUPPORTED_SERVICES`. If an application tries to use a service that is not supported by the service provider, a `ServiceException` is thrown.

This interface contains only synchronous methods. When requesting a service through this interface, the application hands over the control to the service provider. This means that if, for example, an application requests geocoding a street address into coordinates, the geocoding service provider may take over the whole screen from the application and display geocoding information on the screen. The application's execution is blocked until geocoding service has been completed. The application **may** cancel the synchronous service request with `ServiceProvider.reset()` method. This will cause the service request to throw an `InterruptedException`.

Method Summary

`Enumeration`	`geocode(AddressInfo address, boolean dialogsAllowed, Coordinates locationHint)` Requests a geocoding service from a geocoding service provider.
`Enumeration`	`reverseGeocode(Coordinates coordinates, boolean dialogAllowed)` Requests a reverse geocoding service from a geocoding service provider.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceProvider`

`configureProvider, getLanguage, getProviderInfo, getTimeout, reset, setLanguage, setTimeout, supportsArea`

Methods

geocode

```
public Enumeration geocode(AddressInfo address,
                           boolean dialogsAllowed,
                           Coordinates locationHint)
    throws InterruptedException,
           ServiceException
```

(continued from last page)

Requests a geocoding service from a geocoding service provider. This synchronous method blocks until the service request is completed. The returned `Enumeration` contains `Landmark` objects. The application may cancel the synchronous service request with `ServiceProvider.reset()` method. This causes the method to throw an `InterruptedException`.

Since the geocoding converts a street address into coordinates, it is clear that the more complete the address information is, easier it is for a geocoding service provider to find the matching coordinates. Therefore `AddressInfo` provided an an input parameter **should** contain at least a street address.

The geocoding service provider **must** return the best match for the given address info as the first element in the returned `Enumeration`. How the different matches are ranked into order is left to the geocoding service provider. If the geocoding service provider can not find good enough matches for the given address info, it may return an empty `Enumeration`.

If the provided address info is incomplete, the geocoding service provider may find several coordinates for it. This situation can happen, for example, when several cities have the same street name and the given `AddressInfo` does not contain city information. The geocoding service provider may show dialogs to the user to complete the address. The application controls whether dialogs from a service provider are allowed with parameter `dialogsAllowed` in the service request. If the geocoding service provider is allowed to display dialogs and it completes the address info, the `AddressInfo` object passed as a parameter is updated during the service request and it the returned `AddressInfo` object **must** contain all the information geocoding service provider obtained from the user. If the user cancels the dialog, an empty `Enumeration` is returned. If the application wants to make sure that the exact match for the given address info is found, it should allow the dialogs from the geocoding service provider.

With parameter `locationHint` applications can provide a hint for the geocoding service provider where the best match for the service request is. Providing this hint area is optional and `null` indicates that no hint is given.

Parameters:

`address` - the address to convert into coordinates, may be updated during the method call by geocoding service provider
`dialogsAllowed` - `true` if the service provider is allowed to display dialogs to complete the `address`, `false` if dialogs are not allowed
`locationHint` - a hint for the service provider where the best match is, may be `null`

Returns:

an `Enumeration` containing one or more `Landmark` objects for the specified address, an empty `Enumeration`, if no match can be found

Throws:

`java.lang.NullPointerException` - if `address` is `null`
`java.lang.InterruptedException` - if the request has been canceled by the application
`ServiceException` - if the service provider can not serve the request

reverseGeocode

```
public Enumeration reverseGeocode(Coordinates coordinates,
          boolean dialogAllowed)
  throws InterruptedException,
          ServiceException
```

Requests a reverse geocoding service from a geocoding service provider. This synchronous method blocks until the service request is completed. The returned `Enumeration` contains `Landmark` objects. An application may cancel the synchronous service request with `ServiceProvider.reset()` method. This causes the method to throw an `InterruptedException`.

The geocoding provider may want to present some dialogs to the user during reverse geocoding service request. The application controls the whether the dialogs from the service provider are allowed with parameter `dialogsAllowed` in the service request.

In some cases the provided coordinates can be mapped into several street addresses. This situation can happen, for example, if the coordinates are in the street crossing and can be mapped into either of the crossing streets. The service provider **must** return the best match as the first element in the `Enumeration`. How the different matches are ranked into order is left to the service provider. If the geocoding service provider can not find good enough matches for the given coordinates, it may return an empty `Enumeration`

(continued from last page)

Parameters:

`coordinates` - coordinates to be converted to a street address.

`dialogAllowed` - `true` if the service provider is allowed to display dialogs to complete the `address`, `false` if dialogs are not allowed

Returns:

an `Enumeration` containing one or more `Landmark` objects for the specified coordinates, an empty `Enumeration`, if no match can be found

Throws:

`java.lang.NullPointerException` - if `coordinates` is `null`

`java.lang.InterruptedException` - if the request has been canceled by the application

`ServiceException` - if the service provider can not serve the request

com.sirf.microedition.location.services Interface GeographicArea

All Known Implementing Classes:
RectangleGeographicArea, PolygonGeographicArea, CircleGeographicArea

---

public interface **GeographicArea**
extends

This abstract class represents a geographical area in WGS84 coordinate system. It is a superclass for the more shape specific geographic area classes `CircleGeographicArea`, `RectangleGeographicArea` and `PolygonGeographicArea`.

Field Summary

`public static final`	`DOWN` Constant for panning the geographic area downwards Value: **2**
`public static final`	`LEFT` Constant for panning the geographic area to left Value: **4**
`public static final`	`RIGHT` Constant for panning the geographic area to right Value: **8**
`public static final`	`UP` Constant for panning the geographic area upwards Value: **1**

Method Summary

`void`	`centerOn(Coordinates coords)` Centers this area to given coordinates while preserving the scale and orientation of the area.
`boolean`	`containsCoordinates(Coordinates coordinate)` Checks whether the given coordinate is inside this `GeographicArea` object.
`RectangleGeographicArea`	`getBoundingBox(float direction)` Returns a `RectangleGeographicArea` object that surrounds this area compeletely.
`float`	`getDirection()` Returns the direction of the geographic area.
`void`	`pan(int panDirection, int newArea)` Pans the geographic area to the given direction.
`void`	`rescale(int percentage)` With this method an application can modify the scale of the geographic area on the map.
`void`	`rotate(float azimuth)` Rotates the geographic area to the given azimuth.

Fields

UP

```
public static final int UP
```

Constant for panning the geographic area upwards
Constant value: **1**

DOWN

```
public static final int DOWN
```

Constant for panning the geographic area downwards
Constant value: **2**

LEFT

```
public static final int LEFT
```

Constant for panning the geographic area to left
Constant value: **4**

RIGHT

```
public static final int RIGHT
```

Constant for panning the geographic area to right
Constant value: **8**

Methods

rescale

```
public void rescale(int percentage)
  throws ServiceException
```

With this method an application can modify the scale of the geographic area on the map. The new scale is given as percentage of the original size. The rescaling is done to the existing `GeographicArea` object instance.

For example, if the scale of the map is 1:10000 and method `rescale(50)` is called, the resulting scale will be 1:5000 and the map is zoomed in.

Parameters:
`percentage` - percentage of the rescale

Throws:
`java.lang.IllegalArgumentException` - if `percentage` < 1
`ServiceException` - if the `GeographicArea` object can not be rescaled

pan

```
public void pan(int panDirection,
         int newArea)
  throws ServiceException
```

(continued from last page)

Pans the geographic area to the given direction. The `newArea` parameter is the amount of new area in per cents to be included into the geographic area. This parameter is only a request and the API implementation may override it. So if `newArea` is 75, new geographic area contains 75% new area and 25% old area. If the `newArea` > 100%, the new geographic area does not have any old area in it. The `panDirection` **must** be one of the constants defined in this class or a bitwise combination of them. If opposite directions are added to the bitwise combination, those directions are ignored. The pannig is done to the existing `GeographicArea` object instance.

Parameters:

`panDirection` - direction to pan to

`newArea` - the amount of new area after panning in percents

Throws:

`java.lang.IllegalArgumentException` - if `panDirection` is not one of the constants defined in this class or a bitwise combination of them or if `newArea` 1

`ServiceException` - if the `GeographicArea` object can not be panned

---

rotate

```
public void rotate(float azimuth)
  throws ServiceException
```

Rotates the geographic area to the given azimuth. The azimuth is given in degrees relative to true north. Azimuth value is always in the range [0.0, 360.0). The rotating is done to the existing `GeographicArea` object instance.

Parameters:

`azimuth` - the rotation azimuth from true north

Throws:

`java.lang.IllegalArgumentException` - if `angle` < 0 or `angle` 360;

`ServiceException` - if the `GeographicArea` object can not be rotated

---

getDirection

```
public float getDirection()
```

Returns the direction of the geographic area. The value may be the one given as an input parameter in the constructor or value changed with rotate method.

The direction represent the screen up direction on the terminal. For example, when the map image is displayed or created using this object, the service provider **should** generate it that way.

Returns:

the direction of the geographic area

---

containsCoordinates

```
public boolean containsCoordinates(Coordinates coordinate)
  throws ServiceException
```

Checks whether the given coordinate is inside this `GeographicArea` object.

Parameters:

`coordinate` - a coordinates to be checked

Returns:

`true`, if the given point is inside this geographical area, else `false`. If the given coordinate is `null`, `false` is retured.

(continued from last page)

Throws:

`ServiceException` - if the API implementation does not support calculations of containment

getBoundingBox

```
public RectangleGeographicArea getBoundingBox(float direction)
  throws ServiceException
```

Returns a `RectangleGeographicArea` object that surrounds this area compeletely.

Parameters:

`direction` - The requested azimuth for the created bounding box.

Returns:

A RectangleGeographicArea object that surrounds this area.

Throws:

`ServiceException` - if RectangleGeographicArea with requested orientation cannot be created.

centerOn

```
public void centerOn(Coordinates coords)
  throws ServiceException
```

Centers this area to given coordinates while preserving the scale and orientation of the area.

Parameters:

`coords` - Coordinates to center on.

Throws:

`ServiceException` - if this area cannot be centered on the passed coordinates.

com.sirf.microedition.location.services
Class MapLayer

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.MapLayer
```

Direct Known Subclasses:
StaticMapLayerWithCircleArea

public class **MapLayer**
extends Object

This class represents a dynamic map layer generated by a map service provider. With this object, an application can render a map layer on the context it is providing and on top of that whatever it needs to render.

The initial map viewport is set when calling the `MapServiceProvider.getMapLayer` method. After getting a `MapLayer` object, there are two ways to pan and zoom this map object. That can be done either by calling `setGeographicArea` method with a new area, or by using the `pan`, `rotate` and `zoom` methods provided in this class. After any of these operations, the realized new `GeographicArea` object that the current map represents can be retrieved with `getGeographicArea` method.

Some API implementations may retrieve the map data over the network. Therefore the complete map may not be instantly available for rendering. If an application accepts that the map is gradually rendered on the provided graphics context, it may render the map with the content it currently has and register a `MapServiceListener` to get notified when an update to the map data is available.This class also provides two convenience methods that help in converting points on the map image to WGS84 coordinates (see reference [WGS84]) and vice versa.

Method Summary	
`void`	`centerOnCoordinates(Coordinates center)` Centers the map layer to the given coordinates.
`int`	`getCurrentZoomLevel()` Returns the zoom level that is currently used in this map layer.
`GeographicArea`	`getGeographicArea()` Returns the `GeographicArea` object that this map object currently represents.
`int`	`getHeight()` Returns the height of the map layer image as pixels.
`int`	`getWidth()` Returns the width of the map layer image as pixels.
`boolean`	`isValidZoomlevel(int z)` Checks if passed zoom level is valid with this map layer.
`void`	`pan(int x, int y)` Pans the map layer to the given direction.
`void`	`renderMap(Object graphics, boolean needCompleteImage)` Renders this map to the provided graphics context passed in as parameter `graphics`.
`void`	`rotate(int angle)` Rotates the map layer to the given angle.

void	`setGeographicArea(GeographicArea area)` Set a new geographic area for the map layer.
void	`setMapListener(MapServiceListener listener)` Sets a `MapServiceListener` to be called whenever API implementation has received map new data from the network.
int[]	`transformToImageCoordinates(Coordinates coordinates)` Converts coordinates in WGS84 projection system into the coordinate system of the map image.
`Coordinates`	`transformToWGS84Coordinates(int x, int y)` Converts a point on the map image into coordinates in the WGS84 projection.
void	`zoom(int newLevel)` Zooms the map layer to the given zoom level.

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

Methods

renderMap

```
public void renderMap(Object graphics,
          boolean needCompleteImage)
   throws InterruptedException,
          ServiceException
```

Renders this map to the provided graphics context passed in as parameter `graphics`. The API implementation draws a map image in size that was specified when retrieving this `MapLayer` instance. This method blocks until the rending is done.

With parameter `needCompleteImage` an application can control whether the map image rendered to the graphics context is complete or if it can be be updated later when data from the network is available. If an application is able to handle incomplete images, it should register `MapServiceListener` to recieve notifications when new map data is available.

Parameters:

`graphics` - the graphics context the image is rendered to

`needCompleteImage` - `true` if an application must have complete image rendered to the context, `false` if map image may be updated later

Throws:

`java.lang.NullPointerException` - if `graphics` is `null`

`java.lang.InterruptedException` - if the service request has been canceled by the application

`ServiceException` - if rendering is not supported or it fails for some reason

getGeographicArea

```
public GeographicArea getGeographicArea()
```

Returns the `GeographicArea` object that this map object currently represents.

Returns:

a `GeographicArea` object of the map image

setGeographicArea

```
public void setGeographicArea(GeographicArea area)
  throws ServiceException
```

Set a new geographic area for the map layer. The API implementation may do some scaling or panning so that it will be able to render the layer.

Parameters:

`area` - the `GeographicArea` requested for new map area

Throws:

`java.lang.NullPointerException` - if `area` is `null`

`ServiceException` - if some error occurs when setting this new area, for example if this map provider does not support requested area

pan

```
public void pan(int x,
         int y)
  throws ServiceException
```

Pans the map layer to the given direction.

Parameters:

`x` - the number of x pixels to be panned in horizontal direction

`y` - the number of y pixels to be panned in vertical direction

Throws:

`ServiceException` - if the map layer cannot be panned as requested, for example, due to exceeding the supported area of the service provider

rotate

```
public void rotate(int angle)
  throws ServiceException
```

Rotates the map layer to the given angle. The angle is given as degrees from true north.

Parameters:

`angle` - the rotation angle from true north

Throws:

`java.lang.IllegalArgumentException` - if `angle` < 0 or `angle` 360;

`ServiceException` - if the map layer can not be rotated as requested

zoom

```
public void zoom(int newLevel)
  throws ServiceException
```

Zooms the map layer to the given zoom level. The zoom levels supported by the map service provider can be retrieved from the `ProviderInfo.getProperty` with key `MAP_SP_ZOOM_LEVELS`.

Parameters:

`newLevel` - new zoom level to be set

Throws:

`java.lang.IllegalArgumentException` - if the `newLevel` is not one of the values retrieved with MAP_SP_ZOOM_LEVELS key

(continued from last page)

`ServiceException` - if new zoom level can not be set

getCurrentZoomLevel

```
public int getCurrentZoomLevel()
```

Returns the zoom level that is currently used in this map layer. The return value is on of the constants retrieved with `ProviderInfo.getProperty(key)` method where the key value is `MAP_SP_ZOOM_LEVELS`.

Returns:
the current zoom level of the map layer

getWidth

```
public int getWidth()
```

Returns the width of the map layer image as pixels. This is the same value that when requesting this map layer from the service provider with `MapServiceProvider.getMapLayer` method.

Returns:
the width of the map layer in pixels

getHeight

```
public int getHeight()
```

Returns the height of the map layer image as pixels. This is the same value that when requesting this map layer from the service provider with `MapServiceProvider.getMapLayer` method.

Returns:
the height of the map layer in pixels

centerOnCoordinates

```
public void centerOnCoordinates(Coordinates center)
  throws ServiceException
```

Centers the map layer to the given coordinates. The zoom level is not changed in this operation.

Parameters:
`center` - the `Coordinates` to be centered on

Throws:
`ServiceException` - if the map layer can not be centered on given coordinates

setMapListener

```
public void setMapListener(MapServiceListener listener)
```

Sets a `MapServiceListener` to be called whenever API implementation has received map new data from the network.

Parameters:
`listener` - the listener to be notified about map data updates

transformToWGS84Coordinates

```
public Coordinates transformToWGS84Coordinates(int x,
        int y)
```

(continued from last page)

Converts a point on the map image into coordinates in the WGS84 projection. `x` and `y` are presented in the coordinates system of the map image. The origin point is the bottom left corner of the map image. The values on the `x` axis grow horizontally towards the right edge of the map image and on the `y` axis vertically towards to the bottom edge of the map image.

This method does not include any information about the altitude into the returned `Coordinates` object. Therefore the altitude is set to `Float.NaN`.

Parameters:

`x` - position of the point in the horizontal axis of the map image

`y` - position of the point in the vertical axis of the map image

Returns:

the `Coordinates` object for the given point in the WGS84 projection, `null` if conversion can not be done

transformToImageCoordinates

`public int[] transformToImageCoordinates(Coordinates coordinates)`

Converts coordinates in WGS84 projection system into the coordinate system of the map image. The first element in the returned array contains the point on the `x` axis and the second element is the point on the `y` axis of the map image. The origin point is the top left corner of the map image. The values on the `x` axis grow to right and on the `y` axis grow to down.

Parameters:

`coordinates` - coordinates in WGS84 projection to be converted

Returns:

an `int` array containing two items, first the point in the `x` and second the point in the `y` axis in the image coordinate system, `null` if conversion can not be done or the given coordinates are not within in the image

Throws:

`java.lang.NullPointerException` - if `coordinates` is `null`

isValidZoomlevel

`public boolean isValidZoomlevel(int z)`

Checks if passed zoom level is valid with this map layer.

Parameters:

`z` - Zoomlevel to be checked.

Returns:

True, if zoom level is valid, false otherwise.

com.sirf.microedition.location.services
Interface MapServiceListener

All Superinterfaces:
ServiceListener

public interface **MapServiceListener**
extends ServiceListener

This interface provides a callback mechanism for the map service providers to send information about the service request to applications. Applications implement this interface and register it in the asynchronous service request in `MapServiceProvider` to obtain information from the service provider. The relationship with the `MapServiceProvider` methods is descibed in the `MapServiceProvider` class.

Method Summary

`void`	`mapChanged(MapLayer map)` Called by the map service provider when the map layer data has changed, for example due to retrieving new data from network.
`void`	`selectionUpdated(GeographicArea[] area, Landmark[] landmarks, Coordinates[] coordinates)` Called by the map service provider when an application has requested a selection or multiple selections from the map from the user.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceListener`

`requestCanceled`, `requestCompleted`, `requestError`, `requestReset`, `requestTimeout`

Methods

selectionUpdated

```
public void selectionUpdated(GeographicArea[] area,
        Landmark[] landmarks,
        Coordinates[] coordinates)
```

Called by the map service provider when an application has requested a selection or multiple selections from the map from the user. The notification contains information about all currently selected items.

Parameters:

`area` - an array of selected areas, an empty array if no area has been selected

`landmarks` - an array of currently selected `Landmark` objects, an empty array if no landmark has been currently selected

`coordinates` - an array `Coordintates` objects of currently selected points, an empty array if no point has been currently selected

mapChanged

```
public void mapChanged(MapLayer map)
```

Called by the map service provider when the map layer data has changed, for example due to retrieving new data from network. An application **should** refresh its screen after this callback method is called, and call the renderMap(Graphics, boolean) again to get more complete map image rendered.

(continued from last page)

Parameters:

`map` - The map that has changed.

com.sirf.microedition.location.services
Interface MapServiceProvider

All Superinterfaces:
ServiceProvider

public interface **MapServiceProvider**
extends ServiceProvider

This interface collects the services that map providers offer. Through this interface applications are able to request map related services from the map service provider. There are both asynchronous and synchronous methods in this interface.

In asynchronous methods the application hands over the control to the map service provider. This means that the map service provider may take over the whole screen from the application to display maps or select targets from the map. The state and results of the asynchronous service requests are delivered to the application through `MapServiceListener` interface callback methods. In order to receive this results, the application must implement that interface and set the listener in the asynchronous method calls.

The synchronous methods block the execution until the service request has been completed. An application may cancel the synchronous service request with `ServiceProvider.reset()` method. This will cause the service request to throw an `InterruptedException`.

Field Summary

`public static final`	`MAP_TYPE_HYBRID` Map type for hybrid map. Value: **`202`**
`public static final`	`MAP_TYPE_REGULAR` Map type for regular map. Value: **`200`**
`public static final`	`MAP_TYPE_SATELLITE` Map type for satellite map. Value: **`201`**
`public static final`	`SELECT_CIRCLE` Constant for selecting a `CircleGeographicArea` on the map. Value: **`2`**
`public static final`	`SELECT_LANDMARKS` Constant for selecting a `Landmark` object(s) on the map. Value: **`8`**
`public static final`	`SELECT_LOCATION` Constant for selecting a `Location` object on the map. Value: **`16`**
`public static final`	`SELECT_POLYGON` Constant for selecting a `PolygonGeographicArea` on the map. Value: **`4`**
`public static final`	`SELECT_RECTANGLE` Constant for selecting a `RectangleGeographicArea` on the map. Value: **`1`**

Method Summary

void	`displayMap(GeographicArea area, Landmark[] landmarks, Coordinates location, Route route, int mapType, int allowSelection, boolean multipleSelected, boolean saveSelectionOrder, MapServiceListener listener)` Requests a map service provider to display a map.
`MapLayer`	`getMapLayer(GeographicArea area, Landmark[] landmarks, Coordinates location, Route route, int width, int height, int mapType)` Requests a `MapLayer` object from the map service provider.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceProvider`

`configureProvider`, `getLanguage`, `getProviderInfo`, `getTimeout`, `reset`, `setLanguage`, `setTimeout`, `supportsArea`

Fields

SELECT_RECTANGLE

`public static final int` **`SELECT_RECTANGLE`**

Constant for selecting a `RectangleGeographicArea` on the map.
Constant value: **`1`**

SELECT_CIRCLE

`public static final int` **`SELECT_CIRCLE`**

Constant for selecting a `CircleGeographicArea` on the map.
Constant value: **`2`**

SELECT_POLYGON

`public static final int` **`SELECT_POLYGON`**

Constant for selecting a `PolygonGeographicArea` on the map.
Constant value: **`4`**

SELECT_LANDMARKS

`public static final int` **`SELECT_LANDMARKS`**

Constant for selecting a `Landmark` object(s) on the map.
Constant value: **`8`**

SELECT_LOCATION

`public static final int` **`SELECT_LOCATION`**

Constant for selecting a `Location` object on the map.
Constant value: **`16`**

(continued from last page)

MAP_TYPE_REGULAR

```
public static final int MAP_TYPE_REGULAR
```

Map type for regular map. This can be for example a street map or a topographic map
Constant value: **200**

MAP_TYPE_SATELLITE

```
public static final int MAP_TYPE_SATELLITE
```

Map type for satellite map. This can be for example a satellite image or a aerial photograph
Constant value: **201**

MAP_TYPE_HYBRID

```
public static final int MAP_TYPE_HYBRID
```

Map type for hybrid map. Hybrid map combines satellite image or aerial photograph with information from regular map, like borders or roads.
Constant value: **202**

Methods

displayMap

```
public void displayMap(GeographicArea area,
         Landmark[] landmarks,
         Coordinates location,
         Route route,
         int mapType,
         int allowSelection,
         boolean multipleSelected,
         boolean saveSelectionOrder,
         MapServiceListener listener)
  throws ServiceException
```

(continued from last page)

Requests a map service provider to display a map. An application may specify the geographical area, a set of landmarks, a location and a route to be shown on the map. An application may request only one of these alternatives or any combination of them. An application may allow the user to select some items from the map. This is controlled with `allowSelection` parameter in the method call. The value of the `allowSelection` may be one of the constants defined in this interface or a bitwise combination of them.

If an application allows user to select items from the map, it may allow selection on one or multiple items. This is controlled by `multipleSelected` parameter in the method call. If `multipleSelected` is set to `true` an application may select multiple items of the same type or different types is more than one type is selectable. The selection is delivered to the application through `MapServiceListener.selectionUpdated` method. The API implementation **should** call this method only once when the selection has been completed. When the map service provider has completed the UI for the selection a `ServiceListener.requestCompleted()` notification is sent to the application.

If an application requests a geographical area to be shown, the whole area **must** fit to the map shown by the map service provider. If an application requests a location to be shown, that location **must** be shown on the map. If an application requests a geographical area and a location to be shown, both of these **must** fit to the map shown, but the map service provider may do some scaling to fit them both. With landmarks and routes the map service provider may select the size of the map shown and only part of landmarks or route may be visible. If an application provides a location, a set of landmarks or a route in addition to a geographical area to be shown on the map and the specified object is not within the geographical area, at least the geographic area **must** be shown on the map. The map service provider may also provide means for the user to pan or rotate the map shown on the screen.

Possible map type values can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `MAP_SP_SUPPORTED_MAP_TYPES`. Passing `null` as the parameter means that there is no preference for map type.

This is an asynchronous method. The service request is passed through this interface to the a service provider. The status of the service request is notified through the `MapServiceListener` callback methods and this listener is set in the method call. When an application is only showing a map to the user and does not allow any selections to be made from map, the `ServiceListener.requestCompleted` notification is sent to the application when the user exits the map. If an application allows the user to make one selection from the map, the selected items (one or many) are received through `MapServiceListener.selectionUpdated` notification. This notification contains information about all currently selected items. When the user the exits map, `ServiceListener.requestCompleted` notification is sent to the application.

Parameters:

`area` - the geographic area to be shown on the map, may be `null`
`landmarks` - landmarks to be shown on the map, may be `null`
`location` - the location to be shown on the map, may be `null`
`route` - the route to be shown on the map, may be `null`
`mapType` - the type of the map to be shown
`allowSelection` - an element to be selected from the map, or a bitwise combination of selectable items defined in this interface
`multipleSelected` - `true` if the user may select more than one item from the map, else `false`
`saveSelectionOrder` - `true`, if the map service provider must store the order of selected items within one element, else `false`,
`listener` - the listener that receives the status information about the service request

Throws:

`IllegalArgumentException` - if `mapType` is not one of the values retrieved with `ProviderInfo.getProperty` method
`NullPointerException` - if `listener` is `null`
`java.lang.NullPointerException` - if `listener` is `null`
`ServiceException` - if the service provider can not serve the request

(continued from last page)

getMapLayer

```
public MapLayer getMapLayer(GeographicArea area,
         Landmark[] landmarks,
         Coordinates location,
         Route route,
         int width,
         int height,
         int mapType)
  throws InterruptedException,
         ServiceException
```

Requests a `MapLayer` object from the map service provider. An application may specify an area, landmarks,a location and a route that are rendered by the implementation on the generated map image. An application may request only one of these alternatives or any combination of them. If an application requests a geographical area to be selected from, the whole area **must** fit to the generated map image. In other cases, the map service provider may select the map included into the generated map.

The map service provider generates the initial map image on the layer according to the given parameters. Possible map type values can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `MAP_SP_SUPPORTED_MAP_TYPES`. Passing `null` as the parameter means that there is no preference for map type. If the map service provider is not able to perform the request, a `ServiceException` is thrown.

This is a synchronous method and it blocks until the map image generation has been completed. The application may cancel the synchronous service request with `ServiceProvider.reset()` method. This will cause the service request to throw an `InterruptedException`.

Parameters:

`area` - the geographic area to be rendered by implementation on the map, may be `null`
`landmarks` - landmarks to be rendered by implementation on the map, may be `null`
`location` - the location to be rendered by implementation on the map, may be `null`
`route` - route to be included rendered by implementation on the map, may be `null`
`width` - the width of the requested image in pixels
`height` - the height of the requested image in pixels
`mapType` - the type of the map to be generated

Returns:

a generated `MapLayer` object or `null` if the map service provider is not able to generate the map according to the input parameters

Throws:

`java.lang.IllegalArgumentException` - if `width` < 0 or `height` < 0 or if `mapType` is not one of the values retrieved with `ProviderInfo.getProperty` method or if requested map area is not supported by the service provider
`java.lang.InterruptedException` - if the service request has been canceled by the application
`ServiceException` - if the service provider does not support map image generation or if the map can not be generated according to the given parameters

com.sirf.microedition.location.services
Interface NavigationServiceListener

All Superinterfaces:
ServiceListener

public interface **NavigationServiceListener**
extends ServiceListener

This interface provides a callback mechanism for navigation service providers to announce their state and results to applications. Applications implement this interface and register it as input parameter in the asynchronous service request to obtain information about requests.

Method Summary

`void`	`destinationReached()` Called by a navigation service provider when the navigation to the destination has been finished.
`void`	`routeCalculated()` Called by a navigation service provider when it has finished calculating the requested route.
`void`	`routeChanged()` Called by a navigation service provider when it has recalculated the route without explicit request from the application.
`void`	`waypointReached(Coordinates waypoint)` Called by a navigation service provider when a waypoint on the route has been reached.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceListener`

`requestCanceled`, `requestCompleted`, `requestError`, `requestReset`, `requestTimeout`

Methods

routeCalculated

`public void **routeCalculated**()`

Called by a navigation service provider when it has finished calculating the requested route.

waypointReached

`public void **waypointReached**(Coordinates waypoint)`

Called by a navigation service provider when a waypoint on the route has been reached.

Parameters:
`waypoint` - the reached waypoint on the route

destinationReached

`public void **destinationReached**()`

(continued from last page)

Called by a navigation service provider when the navigation to the destination has been finished.

routeChanged

`public void` **`routeChanged`**`()`

Called by a navigation service provider when it has recalculated the route without explicit request from the application. This can happen, for example, when the user goes off route and the navigation service provider notices this and tries to correct it.

com.sirf.microedition.location.services Interface NavigationServiceProvider

All Superinterfaces:

ServiceProvider

public interface **NavigationServiceProvider**
extends ServiceProvider

This interface collects the services that navigation providers offer. Through this interface applications are able to request navigation related services from the navigation service provider. There are both asynchronous and synchronous methods in this interface.

When requesting an asynchronous service through this interface, applications hand over the control to the service provider. This means that if, for example, an application requests navigation to specified location, the navigation provider may take over the whole screen from the application and display the navigation information on the screen. The state and results of the asynchronous service requests are delivered through `NavigationServiceListener` interface callback methods. In order to receive these results, the application must implement that interface and set the listener in the asynchronous method calls.

The synchronous methods block the execution until the service request has been completed. The application may cancel the synchronous service request with `ServiceProvider.reset()` method. This will cause the service request to throw an `InterruptedException`.

Some navigation service providers support generating routes from the specified set of coordinates. This interface provides a mechanism to generate routes. Since generating a route may be an extensive operation, the navigation service provider **is not required** to support generation of more than one route at a time.

Method Summary

`Route[]`	`getRoute(Coordinates[] waypoints, RoutePreferences preferences)` Requests a routing service from a navigation service provider.
`void`	`navigate(Coordinates[] waypoints, RoutePreferences preferences, NavigationServiceListener listener)` Requests a navigation service from a navigation service provider.
`void`	`navigate(Route route, RoutePreferences preferences, NavigationServiceListener listener)` Requests a navigation through the given `Route` object from a navigation service provider.

Methods inherited from interface `com.sirf.microedition.location.services.ServiceProvider`

`configureProvider, getLanguage, getProviderInfo, getTimeout, reset, setLanguage, setTimeout, supportsArea`

Methods

navigate

```
public void navigate(Coordinates[] waypoints,
                     RoutePreferences preferences,
                     NavigationServiceListener listener)
    throws ServiceException
```

(continued from last page)

Requests a navigation service from a navigation service provider. The coordinates in the first position of the array are considered as the starting point and the last item as the destination for the navigation. All the other coordinates in the `waypoints` array are waypoints in the route. The `waypoints` array **must** always contain at least two coordinates, the starting point and the destination point. If the starting point and the destination are the same and there are no waypoints, the navigation service provider decides how to handle this situation. If an application wants to navigate from the current location to some destination, the application has to fetch the current location itself.

An application may include preferences for the navigation service provider to fill the service request. These preferences include, for example, the type of the route or the preferred transport mode for the route. These preferences are captured into the `RoutePreferences` class that is provided as a parameter to the navigation service provider.

This is an asynchronous method. The service request is passed through this interface to the actual navigation service provider. The status of the service request is notified through the `NavigationServiceListener` callback methods and this listener is set in the method call.

If an application wants to use real-time navigation, it should request it with valid `RoutePreferences`. If the real-time navigation creates cost to the user, the navigation service provider **must** inform this to the user.

Parameters:

`waypoints` - the array of coordinates through which the navigation is done

`preferences` - the preferences for the navigation set by the application, `null` if the application does not have any preferences

`listener` - the listener that receives the status information about the service request

Throws:

`java.lang.NullPointerException` - if `waypoints` or `listener` is `null`

`java.lang.IllegalArgumentException` - if the length of parameter `waypoints` is < 2

`ServiceException` - if the service provider does not have needed navigation information or if the service provider can not serve the request

---

navigate

```
public void navigate(Route route,
          RoutePreferences preferences,
          NavigationServiceListener listener)
  throws ServiceException
```

Requests a navigation through the given `Route` object from a navigation service provider.

The application may include preferences for the navigation service provider to fill the service request. These preferences include, for example, the type of the route or the preferred transport mode for the route. These preferences are captured into the `RoutePreferences` class that is provided as a parameter to the navigation service provider.

This is an asynchronous method. The service request is passed through this interface to the actual navigation service provider. The status of the service request is notified through the `NavigationServiceListener` callback methods and this listener is set in the method call.

If an application wants to use real-time navigation, it should request it with valid `RoutePreferences`.

Parameters:

`route` - the route to be navigated

`preferences` - the preferences for the navigation set by the application, `null` if the application does not have any preferences

`listener` - the listener that receives the status information about the service request

Throws:

`java.lang.NullPointerException` - if `route` or `listener` is `null`

`ServiceException` - if the service provider does not have needed navigation information or if the service provider can not serve the request

---

(continued from last page)

getRoute

```
public Route[] getRoute(Coordinates[] waypoints,
                        RoutePreferences preferences)
    throws InterruptedException,
           ServiceException
```

Requests a routing service from a navigation service provider. The provider may return more than one route alternative. The coordinates in the first position of the `waypoints` array are considered as the starting point and the last item as the destination for the route. All the other coordinates in the `waypoints` array are waypoints in the route. The route is generated through the given waypoints in the given order. Splitting a route into smaller route segments is left to the navigation service provider.

The `waypoints` array **must** always contain at least two coordinates, the starting point and the destination point. If the starting point and the destination are the same and there are no waypoints, the navigation service provider decides how to handle this situation. If an application wants to get a route from the current location to some destination, the application has to fetch the current location itself.

This is a synchronous method and it blocks the execution until the service request has been completed. The application may cancel the synchronous service request with `ServiceProvider.reset()` method. This will cause the service request to throw an `InterruptedException`.

An application may include preferences for the navigation service provider to fill the service request. These preferences include, for example, the type of the route or the preferred transport mode for the route. These preferences are captured into the `RoutePreferences` class that is provided as a parameter to the navigation service provider. Navigation service provider **may** ignore these preferences, if it does not support setting the preferences.

Parameters:

`waypoints` - the coordinates through which a route is generated

`preferences` - the preferences for the navigation set by the application, `null` if the application does not have any preferences

Returns:

an array of `Route` object containing generated routes

Throws:

`java.lang.IllegalArgumentException` - if length of `waypoints` array < 2

`java.lang.NullPointerException` - if `waypoints` is `null`

`java.lang.InterruptedException` - if the service request has been canceled by the application

`ServiceException` - if the service provider can not serve the request

com.sirf.microedition.location.services
Class PolygonGeographicArea

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.PolygonGeographicArea
```

All Implemented Interfaces:
GeographicArea

---

public class **PolygonGeographicArea**
extends Object
implements GeographicArea

This class represents a polygonal geographical area in WGS84 coordinate system. This class follows the boundaries definitions in the OpenLS specification (see reference [OpenLS]).

Fields inherited from interface `com.sirf.microedition.location.services.GeographicArea`
`DOWN`, `LEFT`, `RIGHT`, `UP`

Constructor Summary

`public`	`PolygonGeographicArea(Coordinates[] coordinates, float direction)` Constructs a `PolygonGeographicArea` object from the given coordinate points.

Method Summary

`void`	`centerOn(Coordinates coords)` Centers this area to given coordinates while preserving the scale and orientation of the area.
`boolean`	`containsCoordinates(Coordinates coordinate)` Checks whether the given coordinate is inside this `GeographicArea` object, in other words, perfoms a point-in-polygon check.
`RectangleGeographicArea`	`getBoundingBox(float orientation)` Returns a `RectangleGeographicArea` object that surrounds this area compeletely.
`Coordinates`	`getCenterPointCoords()` Returns the coordinates of the center of this area.
`Coordinates[]`	`getCoordinates()` Returns an array of `Coordinates` objects that form the corners of the polygon area.
`float`	`getDirection()` Returns the current direction of the geographic area.
`void`	`pan(int panDirection, int newArea)` Pans the geographic area to the given direction.
`void`	`rescale(int percentage)` With this method an application can modify the scale of the geographic area on the map.

`void`	`rotate(float azimuth)` Rotates the geographic area to the given azimuth.

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

Methods inherited from interface `com.sirf.microedition.location.services.GeographicArea`
`centerOn, containsCoordinates, getBoundingBox, getDirection, pan, rescale, rotate`

Constructors

PolygonGeographicArea

```
public PolygonGeographicArea(Coordinates[] coordinates,
                             float direction)
```

Constructs a `PolygonGeographicArea` object from the given coordinate points. The `direction` specifies what is the orientation of the geographic area, that is, which way is the top of the screen in the resulting map image. The direction is given as degrees from true north. Only simple polygons must be supported. A polygon is called simple if its boundary is described by exactly one closed path that has no self-intersections.

Parameters:

`coordinates` - coordinates that form this polygon.

`direction` - direction of the geographic area on the map, degrees from true north

Throws:

`java.lang.NullPointerException` - if `coordinates` is `null`

`java.lang.IllegalArgumentException` - if `direction` < 0.0 or `direction` 360.0 or coordinates array does contain less than 3 coordinate points.

`ServiceException` - if the `GeographicArea` object can not be created

Methods

getCoordinates

```
public Coordinates[] getCoordinates()
```

Returns an array of `Coordinates` objects that form the corners of the polygon area.

Returns:

the polygonal coordinates of the geographical area

containsCoordinates

```
public boolean containsCoordinates(Coordinates coordinate)
  throws ServiceException
```

Checks whether the given coordinate is inside this `GeographicArea` object, in other words, perfoms a point-in-polygon check.

Parameters:

`coordinate` - a coordinates to be checked

Returns:

(continued from last page)

`true`, if the given point is inside this geographical area, else `false`, or if the given coordinate is `null`, `false` is returned.

Throws:

`ServiceException` - if the API implementation does not support calculations of containment

---

getDirection

```
public float getDirection()
```

Returns the current direction of the geographic area. The value may be the one given as an input parameter in the constructor or value changed with rotate method.

Returns:

the direction of the geographic area

---

pan

```
public void pan(int panDirection,
            int newArea)
  throws ServiceException
```

Pans the geographic area to the given direction. The `newArea` parameter is the amount of new area in per cents to be included into the geographic area. This parameter is only a request and the API implementation may override it. So if `newArea` is 75, new geographic area contains 75% new area and 25% old area. If the `newArea` > 100%, the new geographic area does not have any old area in it. The `panDirection` **must** be one of the constants defined in this class or a bitwise combination of them. If opposite directions are added to the bitwise combination, those directions are ignored. The pannig is done to the existing `GeographicArea` object instance.

Parameters:

`panDirection` - direction to pan to

`newArea` - the amount of new area after panning in percents

Throws:

`java.lang.IllegalArgumentException` - if `panDirection` is not one of the constants defined in this class or a bitwise combination of them or if `newArea` 1

`ServiceException` - if the `GeographicArea` object can not be panned

---

rescale

```
public void rescale(int percentage)
  throws ServiceException
```

With this method an application can modify the scale of the geographic area on the map. The new scale is given as percentage of the original size. The rescaling is done to the existing `GeographicArea` object instance.

If the scale of the map is 1:10000 and method `rescale(50)` is called, the resulting scale will be 1:5000 and the map is zoomed in. Calling this method does not change the size of the map generated using this geographical area object. To change also the map image, an application **must** request new map image from a service provider with the new `GeographicArea` object to see the effect.

Parameters:

`percentage` - percentage of the rescale

Throws:

`java.lang.IllegalArgumentException` - if `percentage` < 0

`ServiceException` - if the `GeographicArea` object can not be rescaled

(continued from last page)

rotate

```
public void rotate(float azimuth)
  throws ServiceException
```

Rotates the geographic area to the given azimuth. The azimuth is given in degrees relative to true north. Azimuth value is always in the range [0.0, 360.0). The rotating is done to the existing `GeographicArea` object instance.

Parameters:
`azimuth` - the rotation azimuth from true north

Throws:
`java.lang.IllegalArgumentException` - if `angle` < 0 or `angle` 360;
`ServiceException` - if the `GeographicArea` object can not be rotated

getCenterPointCoords

```
protected Coordinates getCenterPointCoords()
```

Returns the coordinates of the center of this area.

Returns:
Center coordinates, null if it cannot be calculated.

getBoundingBox

```
public RectangleGeographicArea getBoundingBox(float orientation)
  throws ServiceException
```

Returns a `RectangleGeographicArea` object that surrounds this area compeletely.

Parameters:
`direction` - The requested azimuth for the created bounding box.

Returns:
A RectangleGeographicArea object that surrounds this area.

Throws:
`ServiceException` - if RectangleGeographicArea with requested orientation cannot be created.

centerOn

```
public void centerOn(Coordinates coords)
  throws ServiceException
```

Centers this area to given coordinates while preserving the scale and orientation of the area.

Parameters:
`coords` - Coordinates to center on.

Throws:
`ServiceException` - if this area cannot be centered on the passed coordinates.

com.sirf.microedition.location.services
Class ProviderInfo

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.ProviderInfo
```

public class **ProviderInfo**
extends Object

This interface collects the information about the capabilities of a service provider. This information includes the type of the service, name of the service provider and additional information about the provider. With this information the application is able to decide which service provider to use when requesting services.

Since the capabilities of the different service providers vary, this interface offers a mechanism to store the capabilities as key and value pairs. For this reason, the interface defines a set of key values for the capabilities that were identified when writing this specification. The API implementation and the service provider **may** add new property keys for new capabilities. In this case the key name **must** include a reverse domain name of the API implementation or the service provider.

If a service provider is able to provider more than one kind of service, for example, map and navigation services, the associated `ProviderInfo` object **must** contain information for all of the supported services.

Field Summary

`public static final`	`COM_SIRF_STUDIO_MAP_LAYERTYPE` Value: **`com.sirf.studio.map.layertype`**
`public static final`	`COM_SIRF_STUDIO_MAP_LAYERTYPE_STATIC` Value: **`static`**
`public static final`	`COM_SIRF_STUDIO_MAP_LAYERTYPE_TILING` Value: **`tiling`**
`public static final`	`COM_SIRF_STUDIO_MAP_PROJECTION` Value: **`com.sirf.studio.map.projection`**
`public static final`	`COM_SIRF_STUDIO_MAP_PROJECTION_LINEAR` Value: **`linear`**
`public static final`	`COM_SIRF_STUDIO_MAP_PROJECTION_MERCATOR` Value: **`mercator`**
`public static final`	`COM_SIRF_STUDIO_MAP_TILED_CONTENT_VERSION` Value: **`com.sirf.studio.map.tiled.version`**
`public static final`	`COM_SIRF_STUDIO_MAP_TILED_TILESIZE` Value: **`com.sirf.studio.map.tiled.tilesize`**
`public static final`	`COM_SIRF_STUDIO_MAP_TILED_ZOOMLEVELS` Value: **`com.sirf.studio.map.tiled.zoom`**

`public static final`	COM_SIRF_STUDIO_NAV_MAPPROV_ID Value: **`com.sirf.studio.nav.mapprov.id`**
`public static final`	COM_SIRF_STUDIO_SP_DEFAULT_VIEWPORT Value: **`com.sirf.studio.sp.default.viewport`**
`public static final`	COM_SIRF_STUDIO_SP_ID Value: **`com.sirf.studio.sp.id`**
`public static final`	COM_SIRF_STUDIO_SP_NAME Value: **`com.sirf.studio.sp.name`**
`public static final`	COM_SIRF_STUDIO_SP_SUPPORTED_GEOGRAPHIC_AREA Value: **`com.sirf.studio.sp.supported_areas`**
`public static final`	COM_SIRF_STUDIO_SP_URL Value: **`com.sirf.studio.sp.url`**
`public static final`	CONFIGURATION_UI Property key for a service provider to determine whether is has a separate configuration UI that can be shown to the user. Value: **`configuration_ui`**
`public static final`	CREATE_NETWORK_TRAFFIC Property key value for a service provider to determine whether it creates network traffic when performing the service request. Value: **`create_network_traffic`**
`public static final`	GEO_SP_SUPPORTED_GEOCODING Property key value for a geocoding service provider to determine whether it supports geocoding service. Value: **`geo_sp_supportes_geocoding`**
`public static final`	GEO_SP_SUPPORTED_REVERSE_GEOCODING Property key value for a geocoding service provider to determine whether it supports reverse geocoding service. Value: **`geo_sp_supportes_reverse_geocoding`**
`public static final`	MAP_SP_RETURN_MAP_OBJECTS Property key value for a map service provider to determine whether it is able create map images. Value: **`map_sp_return_map_objects`**
`public static final`	MAP_SP_SUPPORTED_MAP_TYPES Property key value for a map service provider to determine what are the map types the provider supports. Value: **`map_sp_supported_map_types`**
`public static final`	MAP_SP_ZOOM_LEVELS Property key value for a map service provider to determine how many zoom levels it supports. Value: **`map_sp_zoom_levels`**

`public static final`	`NAV_SP_GENERATE_ROUTES` Property key value for a navigation service provider to determine whether it support generation of routes. Value: **`nav_sp_generate_routes`**
`public static final`	`NAV_SP_SUPPORTED_AVOID_FEATURES` Property key value for a navigation service provider to specify what avoid features it supports. Value: **`nav_sp_supported_avoid_features`**
`public static final`	`NAV_SP_SUPPORTED_NAVIGATION_TYPES` Property key value for a navigation service provider to specify what kind of navigation types it supports. Value: **`nav_sp_supported_navigation_types`**
`public static final`	`NAV_SP_SUPPORTED_ROUTE_TYPES` Property key value for a navigation service provider to specify what kind of route types it supports. Value: **`nav_sp_supported_route_types`**
`public static final`	`NAV_SP_SUPPORTED_TRANSPORT_MODES` Property key value for a navigation service provider to specify what transport modes it supports. Value: **`nav_sp_supported_transport_modes`**
`public static final`	`SERVICE_FEE_ATTACHED` Property key value for a service provider to determine whether it needs a separate subscription from the user. Value: **`service_fee_attached`**
`public static final`	`SUPPORTED_LANGUAGES` Property key value for a service provider to specify what languages it supports. Value: **`supported_languages`**

Constructor Summary

`public`	`ProviderInfo(int providerID, String name, String providerURL, int type, Vector keys, Vector values)`
`public`	`ProviderInfo(int type, Vector keys, Vector values)`

Method Summary

`String`	`getName()` Returns the name of the service provider.
`Object`	`getProperty(String key)` Returns the value of the requested property as an `object`.
`String[]`	`getPropertyKeys()` Returns the list of the property keys that have been set.
`int`	`getType()` Returns the type of the service provider.

boolean	`supportsArea(GeographicArea area)` With this method an application may check if a service provider has a information about the specified geographical area.

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

Fields

SUPPORTED_LANGUAGES

`public static final java.lang.String` **`SUPPORTED_LANGUAGES`**

Property key value for a service provider to specify what languages it supports. The value of this property is presented as `String[]` that contains an array of supported languages as language tags defined in RFC 4646 (see reference [RFC4646]).
Constant value: **`supported_languages`**

CREATE_NETWORK_TRAFFIC

`public static final java.lang.String` **`CREATE_NETWORK_TRAFFIC`**

Property key value for a service provider to determine whether it creates network traffic when performing the service request. Network traffic is needed if, for example, a navigation service provider fetches the maps from the server using data connection. Use of network traffic may create financial cost for the user. The value of this property is presented as `Boolean`.
Constant value: **`create_network_traffic`**

SERVICE_FEE_ATTACHED

`public static final java.lang.String` **`SERVICE_FEE_ATTACHED`**

Property key value for a service provider to determine whether it needs a separate subscription from the user. The value of this property is presented as `Boolean`.
Constant value: **`service_fee_attached`**

CONFIGURATION_UI

`public static final java.lang.String` **`CONFIGURATION_UI`**

Property key for a service provider to determine whether is has a separate configuration UI that can be shown to the user. The value of this property is presented as `Boolean`.
Constant value: **`configuration_ui`**

GEO_SP_SUPPORTED_GEOCODING

`public static final java.lang.String` **`GEO_SP_SUPPORTED_GEOCODING`**

Property key value for a geocoding service provider to determine whether it supports geocoding service. The value of this property is presented as `Boolean`.
Constant value: **`geo_sp_supportes_geocoding`**

GEO_SP_SUPPORTED_REVERSE_GEOCODING

`public static final java.lang.String` **`GEO_SP_SUPPORTED_REVERSE_GEOCODING`**

(continued from last page)

Property key value for a geocoding service provider to determine whether it supports reverse geocoding service. The value of this property is presented as `Boolean`.
Constant value: **`geo_sp_supportes_reverse_geocoding`**

MAP_SP_RETURN_MAP_OBJECTS

public static final java.lang.String **MAP_SP_RETURN_MAP_OBJECTS**

Property key value for a map service provider to determine whether it is able create map images. The value of this property is presented as `Boolean`.
Constant value: **`map_sp_return_map_objects`**

MAP_SP_SUPPORTED_MAP_TYPES

public static final java.lang.String **MAP_SP_SUPPORTED_MAP_TYPES**

Property key value for a map service provider to determine what are the map types the provider supports. The value of this property is presented as `String[]` that contains an array of supported map types. The possible map types are presented with constants defined in `MapServiceProvider` interface and they start with prefix `MAP_TYPE_`.
Constant value: **`map_sp_supported_map_types`**

MAP_SP_ZOOM_LEVELS

public static final java.lang.String **MAP_SP_ZOOM_LEVELS**

Property key value for a map service provider to determine how many zoom levels it supports. The zoom levels start from 1 and run in steps of one to the given number of zoom levels. The value of this property is presented as `Integer`.
Constant value: **`map_sp_zoom_levels`**

NAV_SP_GENERATE_ROUTES

public static final java.lang.String **NAV_SP_GENERATE_ROUTES**

Property key value for a navigation service provider to determine whether it support generation of routes. The value of this property is presented as `Boolean`.
Constant value: **`nav_sp_generate_routes`**

NAV_SP_SUPPORTED_NAVIGATION_TYPES

public static final java.lang.String **NAV_SP_SUPPORTED_NAVIGATION_TYPES**

Property key value for a navigation service provider to specify what kind of navigation types it supports. The value of this property is presented as `String[]` that contains an array of supported navigation types. Some of the navigation types are defined as constants in `RoutePreferences` class and they start with prefix `NAVIGATION_TYPE_`. The list **may** contain also other navigation types.
Constant value: **`nav_sp_supported_navigation_types`**

NAV_SP_SUPPORTED_ROUTE_TYPES

public static final java.lang.String **NAV_SP_SUPPORTED_ROUTE_TYPES**

Property key value for a navigation service provider to specify what kind of route types it supports. The value of this property is presented as `String[]` that contains an array of supported route types. Some of the basic route types are defined as constants in `RoutePreferences` class and they start with prefix `ROUTE_TYPE_`. The list **may** contain also other route types.
Constant value: **`nav_sp_supported_route_types`**

NAV_SP_SUPPORTED_TRANSPORT_MODES

public static final java.lang.String **NAV_SP_SUPPORTED_TRANSPORT_MODES**

(continued from last page)

Property key value for a navigation service provider to specify what transport modes it supports. The value of this property is presented as `String[]` that contains an array of supported transport modes. Some of the basic transport modes are defined as constants in `RoutePreferences` class and they start with prefix `TRANSPORT_`. The list **may** contain also other transportation modes.
Constant value: **`nav_sp_supported_transport_modes`**

NAV_SP_SUPPORTED_AVOID_FEATURES

`public static final java.lang.String` **`NAV_SP_SUPPORTED_AVOID_FEATURES`**

Property key value for a navigation service provider to specify what avoid features it supports. For example toll ways and bridge are avoid features. The value of this property is presented as `String[]` that contains an array of avoid features.

These avoid features must to be returned in the language set for the service provider and **may** be shown to the user. If the application changes the language of the service provider, these names of the avoid feature must to be changed as well.

Some navigation service provider may be able to determine whether traveling routes with different transport modes creates cost to the user. Cost may be a toll road or a bridge fee. If navigation service providers are able to determine cost, they should define cost as one of the avoid features. This way an application may request routes that do not create any cost to the user.
Constant value: **`nav_sp_supported_avoid_features`**

COM_SIRF_STUDIO_SP_NAME

`public static final java.lang.String` **`COM_SIRF_STUDIO_SP_NAME`**

Constant value: **`com.sirf.studio.sp.name`**

COM_SIRF_STUDIO_SP_ID

`public static final java.lang.String` **`COM_SIRF_STUDIO_SP_ID`**

Constant value: **`com.sirf.studio.sp.id`**

COM_SIRF_STUDIO_SP_URL

`public static final java.lang.String` **`COM_SIRF_STUDIO_SP_URL`**

Constant value: **`com.sirf.studio.sp.url`**

COM_SIRF_STUDIO_SP_SUPPORTED_GEOGRAPHIC_AREA

`public static final java.lang.String` **`COM_SIRF_STUDIO_SP_SUPPORTED_GEOGRAPHIC_AREA`**

Constant value: **`com.sirf.studio.sp.supported_areas`**

COM_SIRF_STUDIO_SP_DEFAULT_VIEWPORT

`public static final java.lang.String` **`COM_SIRF_STUDIO_SP_DEFAULT_VIEWPORT`**

Constant value: **`com.sirf.studio.sp.default.viewport`**

(continued from last page)

COM_SIRF_STUDIO_MAP_LAYERTYPE

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_LAYERTYPE`**

Constant value: **`com.sirf.studio.map.layertype`**

COM_SIRF_STUDIO_MAP_PROJECTION

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_PROJECTION`**

Constant value: **`com.sirf.studio.map.projection`**

COM_SIRF_STUDIO_MAP_TILED_CONTENT_VERSION

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_TILED_CONTENT_VERSION`**

Constant value: **`com.sirf.studio.map.tiled.version`**

COM_SIRF_STUDIO_MAP_TILED_ZOOMLEVELS

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_TILED_ZOOMLEVELS`**

Constant value: **`com.sirf.studio.map.tiled.zoom`**

COM_SIRF_STUDIO_MAP_TILED_TILESIZE

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_TILED_TILESIZE`**

Constant value: **`com.sirf.studio.map.tiled.tilesize`**

COM_SIRF_STUDIO_MAP_LAYERTYPE_TILING

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_LAYERTYPE_TILING`**

Constant value: **`tiling`**

COM_SIRF_STUDIO_MAP_LAYERTYPE_STATIC

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_LAYERTYPE_STATIC`**

Constant value: **`static`**

COM_SIRF_STUDIO_MAP_PROJECTION_MERCATOR

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_PROJECTION_MERCATOR`**

Constant value: **`mercator`**

COM_SIRF_STUDIO_MAP_PROJECTION_LINEAR

`public static final java.lang.String` **`COM_SIRF_STUDIO_MAP_PROJECTION_LINEAR`**

(continued from last page)

Constant value: **`linear`**

COM_SIRF_STUDIO_NAV_MAPPROV_ID

```
public static final java.lang.String COM_SIRF_STUDIO_NAV_MAPPROV_ID
```

Constant value: **`com.sirf.studio.nav.mapprov.id`**

Constructors

ProviderInfo

```
public ProviderInfo(int providerID,
                    String name,
                    String providerURL,
                    int type,
                    Vector keys,
                    Vector values)
```

ProviderInfo

```
public ProviderInfo(int type,
                    Vector keys,
                    Vector values)
```

Methods

getType

```
public int getType()
```

Returns the type of the service provider. If the provider supports more than one kind of services, a bitwise combination (OR) of those types is returned. The types are defined as constants in the `ProviderManager` class.

Returns:
the types of the services the provider supports

getName

```
public String getName()
```

Returns the name of the service provider. This name should be descriptive, since the application may base its selection to this name.

Returns:
the name of the service provider

getProperty

```
public Object getProperty(String key)
```

Returns the value of the requested property as an `object`. Returned value is the value set by the API implementation.

(continued from last page)

Parameters:

`key` - the identifier of the property

Returns:

value of the property, `null` if the `key` is not defined

getPropertyKeys

`public String[] getPropertyKeys()`

Returns the list of the property keys that have been set. The actual property value can be retrieved with method `getProperty(key)` . These properties may also be used to set additional route preferences in `RoutePreferences` class.

Returns:

key values of assigned properties

supportsArea

`public boolean supportsArea(GeographicArea area)`

With this method an application may check if a service provider has a information about the specified geographical area. The information may be geocoding data, maps or navigation information depending on the type of the service provider. With this method the check can be made before connecting to any particular service provider.

Parameters:

`area` - geographical area from which the information is needed

Returns:

`true` if the service provider has a information for the requested geographical area, else `false`

Throws:

`java.lang.NullPointerException` - if `area` is `null`

com.sirf.microedition.location.services
Class ProviderManager

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.ProviderManager
```

public class **ProviderManager**
extends Object

This class is the starting point for using service provider interfaces in Location API 2.0. This class is used to search different service providers and get the actual provider instances. First an application searches for a certain service provider and gets back a list from possible providers. Then the application selects which service provider to use and connects to that particular service provider. After this the services from the provider are available for the application to use. A code example how this is done can be found in Appendix B: Code examples.

The different service provider types that an application can connect to are defined as constants in this class. The constants have bit field values. This means that an application can combine service types when searching for a service provider. With the combined service provider type the search result cover service providers for all the combined service types.

If there are several service providers for a service, the API implementation **must** select one of those to be the default service provider. That default service provider is used when an application requests a service provider instance without specifying the name of the provider. For more information on this, see the descriptions in `connectToServiceProvider` methods.

Field Summary

`public static final`	`GEOCODING` Service provider type for geocoding and reverse geocoding services Value: **4**
`public static final`	`MAP` Service provider type for map services Value: **1**
`public static final`	`NAVIGATION` Service provider type for navigation services Value: **2**

Method Summary

`static ServiceProvider`	`connectToServiceProvider(ProviderInfo provider, String extraInfo)` Gets an instance of requested service provider to be used in the service requests.
`static ServiceProvider`	`connectToServiceProvider(String name, int type, String extraInfo)` Gets an instance of requested service provider to be used in the service requests.
`static ProviderInfo[]`	`findServiceProviders(int type, String extraInfo)` With this method the application is able to find service providers that support specified service type.
`static ProviderInfo`	`getProviderInfo(int providerID, int providerType)` Method to find ProviderInfo for given ID and type.
`static void`	`retrieveProviderInfos(String extraInfo)`

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

Fields

MAP

```
public static final int MAP
```

Service provider type for map services
Constant value: **1**

NAVIGATION

```
public static final int NAVIGATION
```

Service provider type for navigation services
Constant value: **2**

GEOCODING

```
public static final int GEOCODING
```

Service provider type for geocoding and reverse geocoding services
Constant value: **4**

Methods

findServiceProviders

```
public static ProviderInfo[] findServiceProviders(int type,
         String extraInfo)
  throws ServiceException
```

(continued from last page)

With this method the application is able to find service providers that support specified service type. The possible type values are defined as constants in this class. This method may return information about more than one service provider. The application can check the properties of different service provides from the returned `ProviderInfo` objects.

An application can search for a service provider that provides more than one type of services. This is done by combining the different type fields with bitwise AND-operation. For example if an application wants to find a service provider that supports both mapping and navigation, the input value for the method is 0x00000003 (=`MAP && NAVIGATION`). If an application requests multiple services, the returned `ProviderInfo` object **must** contain information about all of the requested services.

This method returns an array of `ProviderInfo` objects of all service providers on the device that provide the requested service. The default service provider defined by the API implementation is returned in the first element of the array. If a service provider is able to provide more than one kind of services, the returned `ProviderInfo` object **must** contain information about all of provided services. If an application wants to use this kind of service provider , it **must** connect to these different services separately. That is use `ProviderManager.connectToServiceProvider("myService", ServiceProvider.MAP)` method call to get an instance of `MapServiceProvider`.

In some implementations the capabilities of the service provider may be fetched from a remote server. In this case the execution of this method may take a while, since this methods blocks the execution.

Searching a service providers may require authentication or identifying the application doing the search. This information can be given with `extraInfo` parameter. The format of this `String` depends on the implementation and on the service provider. `null` is used if there is no additional information.

Parameters:

`type` - the type of the requested service, may be a bitwise combination of several types

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

an array `ProviderInfo` objects for all the service providers that support the specified service type, an empty array if there are no providers for the specified service type

Throws:

`java.lang.IllegalArgumentException` - if type is not one of the constants defined in this class or a bitwise combination (AND) of them
`ServiceException` - if service request fails, for example, if information about services can not be fetched from remote server

connectToServiceProvider

```
public static ServiceProvider connectToServiceProvider(String name,
        int type,
        String extraInfo)
```

Gets an instance of requested service provider to be used in the service requests. The type parameter is passed in to clarify which type of service is requested, if the service provider supports more than one type of services. If `null` is given as the service name, the API implementation **must** return the default provider for the requested service type. The type of the service **must** be one of the constant defined in this class or a bitwise combination (AND) of them.

If an application searches for a service provider with combined service types, it may still connect to only one kind of service at a time. This means that even though the `ProviderInfo` object returned from the `findServiceProviders` method contains information about more than one kind of service, the application must select only one service to be used in this method call and cast the general `ServiceProvider` interface into one specific service type class in order to be able to use the specific services defined for that service type.

In some cases the service provider may require authentication or some other additional information. This information can be sent to the service provider with `extraInfo` parameter. The service provider specifies the format of this additional information. There may be service provider specific property keys in the `ProviderInfo` class to define this information. `null` is used if there is no additional information.

Parameters:

`name` - the name of the service provider to be instantiated, `null` to request default provider

(continued from last page)

`type` - the type of the requested service

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

the instance of the requested service provider, `null` if the API implementation does not support requested service type

Throws:

`java.lang.IllegalArgumentException` - if `type` is not one of the constant defined in this class or a bitwise combination (AND) of them

connectToServiceProvider

```
public static ServiceProvider connectToServiceProvider(ProviderInfo provider,
        String extraInfo)
```

Gets an instance of requested service provider to be used in the service requests.

If an application searches for a service provider with combined service types, it may still connect to only one kind of service at a time. This means that even though the `ProviderInfo` object returned from the `findServiceProviders` method contains information about more than one kind of service, the application must select only one service to be used in this method call and cast the general `ServiceProvider` interface into one specific service type class in order to be able to use the specific services defined for that service type.

In some cases the service provider may require authentication or some other additional information. This information can be sent to the service provider with `extraInfo` parameter. The service provider specifies the format of this additional information. There may be service provider specific property keys in the `ProviderInfo` class to define this information. `null` is used if there is no additional information.

Parameters:

`provider` - the service provider to be instantiated

`extraInfo` - additional information, for example, username or URL that is sent to the service provider, `null` if no additional information is needed

Returns:

the instance of the requested service provider, `null` if the API implementation does not support requested service type

Throws:

`java.lang.NullPointerException` - if `provider` is `null`

retrieveProviderInfos

```
protected static void retrieveProviderInfos(String extraInfo)
  throws ServiceException
```

getProviderInfo

```
protected static ProviderInfo getProviderInfo(int providerID,
        int providerType)
```

Method to find ProviderInfo for given ID and type.

Parameters:

`providerID`

`providerType`

Returns:

ProviderInfo, null if not found.

com.sirf.microedition.location.services
Class RectangleGeographicArea

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.RectangleGeographicArea
```

All Implemented Interfaces:
GeographicArea

---

public class **RectangleGeographicArea**
extends Object
implements GeographicArea

This class represents a rectangular geographical area in WGS84 coordinate system.

Fields inherited from interface `com.sirf.microedition.location.services.GeographicArea`
`DOWN`, `LEFT`, `RIGHT`, `UP`

Constructor Summary

`public`	`RectangleGeographicArea(Coordinates cornerPoint, Coordinates oppositeCornerPoint)` Constructs a `RectangleGeographicArea` object.

Method Summary

`void`	`centerOn(Coordinates coords)` Centers this area to given coordinates while preserving the scale and orientation of the area.
`boolean`	`containsCoordinates(Coordinates coordinate)` Checks whether the given coordinate is inside this `GeographicArea` object.
`RectangleGeographicArea`	`getBoundingBox(float orientation)` Returns a `RectangleGeographicArea` object that surrounds this area compeletely.
`Coordinates`	`getCenterPointCoords()`
`Coordinates[]`	`getCoordinates()` Returns an array of `Coordinates` objects that form the rectangle representation of the geographical area.
`float`	`getDirection()` Returns the direction of the geographic area on the map.
`void`	`pan(int panDirection, int newArea)` Pans the geographic area to the given direction.
`void`	`rescale(int percentage)` With this method an application can modify the scale of the geographic area on the map.
`void`	`rotate(float azimuth)` Rotates the geographic area to the given azimuth.

String	toString()

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `com.sirf.microedition.location.services.GeographicArea`

`centerOn, containsCoordinates, getBoundingBox, getDirection, pan, rescale, rotate`

Constructors

RectangleGeographicArea

```
public RectangleGeographicArea(Coordinates cornerPoint,
                               Coordinates oppositeCornerPoint)
```

Constructs a `RectangleGeographicArea` object. The object is created as a rectangle from the `cornerPoint` to the `oppositeCornerPoint` coordinates and the direction is to true north. These coordinates defines opposite corners in arbitrary dimensions.

Parameters:

`cornerPoint` - coordinates of one corner of the geographical area

`oppositeCornerPoint` - coordinates of the opposite corner of the geographical area

Throws:

`java.lang.NullPointerException` - if `cornerPoint` is `null` or `oppositeCornerPoint` is `null` // * @throws java.lang.IllegalArgumentException if the latitude value of`cornerPoint` equals the latitude value of`oppositeCornerPoint` // * or if the longitude value of`cornerPoint` equals the longitude value of`oppositeCornerPoint`

`ServiceException` - if the `GeographicArea` object can not be created

Methods

containsCoordinates

```
public boolean containsCoordinates(Coordinates coordinate)
```

Checks whether the given coordinate is inside this `GeographicArea` object. If the given coordinate is `null`, `false` is returned.

Parameters:

`coordinate` - a coordinates to be checked

Returns:

`true`, if the given point is inside this geographical area, else `false`

Throws:

`ServiceException` - if the API implementation does not support calculations of containment

getDirection

```
public float getDirection()
```

Returns the direction of the geographic area on the map. The value may be the one given as an input parameter in the constructor or value changed with rotate method.

(continued from last page)

Returns:
the direction of the geographic area

pan

```
public void pan(int panDirection,
            int newArea)
  throws ServiceException
```

Pans the geographic area to the given direction. The `newArea` parameter is the amount of new area in per cents to be included into the geographic area. This parameter is only a request and the API implementation may override it. So if `newArea` is 75, new geographic area contains 75% new area and 25% old area. If the `newArea` > 100%, the new geographic area does not have any old area in it. The `panDirection` **must** be one of the constants defined in this class or a bitwise combination of them. If opposite directions are added to the bitwise combination, those directions are ignored. The pannig is done to the existing `GeographicArea` object instance.

Parameters:
`panDirection` - direction to pan to
`newArea` - the amount of new area after panning in percents

Throws:
`java.lang.IllegalArgumentException` - if `panDirection` is not one of the constants defined in this class or a bitwise combination of them or if `newArea` 1
`ServiceException` - if the `GeographicArea` object can not be panned

rescale

```
public void rescale(int percentage)
  throws ServiceException
```

With this method an application can modify the scale of the geographic area on the map. The new scale is given as percentage of the original size. The rescaling is done to the existing `GeographicArea` object instance.

For example, if the scale of the map is 1:10000 and method `rescale(50)` is called, the resulting scale will be 1:5000 and the map is zoomed in.

Parameters:
`percentage` - percentage of the rescale

Throws:
`java.lang.IllegalArgumentException` - if `percentage` < 1
`ServiceException` - if the `GeographicArea` object can not be rescaled

rotate

```
public void rotate(float azimuth)
  throws ServiceException
```

Rotates the geographic area to the given azimuth. The azimuth is given in degrees relative to true north. Azimuth value is always in the range [0.0, 360.0). The rotating is done to the existing `GeographicArea` object instance.

Parameters:
`azimuth` - the rotation azimuth from true north

Throws:
`java.lang.IllegalArgumentException` - if `angle` < 0 or `angle` 360;

(continued from last page)

`ServiceException` - if the `GeographicArea` object can not be rotated

getCoordinates

```
public Coordinates[] getCoordinates()
```

Returns an array of `Coordinates` objects that form the rectangle representation of the geographical area. Open: How to handle crossing 180

Returns:
array containing the bottom left corner coordinates and the top right corner coordinates of this geographical area.

toString

```
public String toString()
```

getBoundingBox

```
public RectangleGeographicArea getBoundingBox(float orientation)
  throws ServiceException
```

Returns a `RectangleGeographicArea` object that surrounds this area compeletely.

Parameters:
`direction` - The requested azimuth for the created bounding box.

Returns:
A RectangleGeographicArea object that surrounds this area.

Throws:
`ServiceException` - if RectangleGeographicArea with requested orientation cannot be created.

centerOn

```
public void centerOn(Coordinates coords)
  throws ServiceException
```

Centers this area to given coordinates while preserving the scale and orientation of the area.

Parameters:
`coords` - Coordinates to center on.

Throws:
`ServiceException` - if this area cannot be centered on the passed coordinates.

getCenterPointCoords

```
protected Coordinates getCenterPointCoords()
```

com.sirf.microedition.location.services
Class Route

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.Route
```

public class **Route**
extends Object

This class represents a route. A route consists of one or more route segments combined in the course of travel. A route has a summary, length and estimated traveling time. A `Route` object also contains an array of `RouteSegment` objects that form the actual route.

Field Summary

`public`	`mapImage`

Constructor Summary

`public`	`Route(RouteSegment[] segments)` Constructor that generates a `Route` object with the given route segments.

Method Summary

`Coordinates`	`getDestinationPoint()` Returns the coordinates of the destination point of the route.
`GeographicArea`	`getGeographicArea(float direction)` Returns the `GeographicArea` of the route.
`double`	`getLength()` Returns the length of the route in meters.
`String`	`getRouteType()` Returns the type of the route that is used in the route calculations by the navigation service provider.
`RouteSegment[]`	`getSegments()` Returns the segments of the route.
`Coordinates`	`getStartingPoint()` Returns the coordinates of the starting point of the route.
`String`	`getSummary()` Returns the summary of the route.
`long`	`getTravelTime()` Returns the estimated time it takes to travel the route.
`void`	`setRouteType(String routeType)` Sets the type of the route.

`void`	`setSegments(RouteSegment[] segments)` Sets the segments for the route.
`void`	`setSummary(String summary)` Sets the summary for the route.

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

Fields

mapImage

`public StaticMapImage` **`mapImage`**

Constructors

Route

`public` **`Route`**`(RouteSegment[] segments)`

Constructor that generates a `Route` object with the given route segments. A `Route` **must** contain at least one segment. The first coordinate in the first segment is the starting point and the last coordinate in the last segment is the destination point of the route.

Parameters:

`segments` - the segments of the route

Throws:

`java.lang.NullPointerException` - if `segments` is `null`

`java.lang.IllegalArgumentException` - if `segments` is empty

Methods

getLength

`public double` **`getLength`**`()`

Returns the length of the route in meters. This length **must** be the sum of lengths of the route segments. The length is given in meters. If this object is created by the application, the API implementation **must** calculate the length from the lenghts of the route segments.

Returns:

the route length in meters

getSegments

`public RouteSegment[]` **`getSegments`**`()`

Returns the segments of the route. The segments **must** be ordered from the starting point to the destination so that they form a continuous route. The `RouteSegment` objects in the returned `Enumeration` contain more information about the segment, like geometry of the segment. A route **must** contain at least one segment. The order of the segments is significant.

(continued from last page)

Returns:
an `Enumeration` containing `RouteSegment` objects for the route segments

setSegments

`public void **setSegments**(RouteSegment[] segments)`

Sets the segments for the route. This method overrides the previously set route segments. If `segments` is `null`, the method returns without any actions. The order of the segments is significant.

Parameters:
`segments` - the route segments of the route

Throws:
`java.lang.NullPointerException` - if `segments` is `null`
`java.lang.IllegalArgumentException` - if `segments` is empty

getSummary

`public String **getSummary**()`

Returns the summary of the route. This summary describes the route in general. This summary may be shown to the user. The `RouteSegment` objects in the route contain descriptions about the segments.

Returns:
route summary, `null` is no summary is available

setSummary

`public void **setSummary**(String summary)`

Sets the summary for the route. The summary describes the route in general. This summary may be shown to the user. The `RouteSegment` objects in the route contain descriptions about the segments.

Parameters:
`summary` - the route summary

getTravelTime

`public long **getTravelTime**()`

Returns the estimated time it takes to travel the route. The return value **must** be the sum of travel times of the route segments. The time is given is seconds. If this object is created by the application, the API implementation **must** calculate the travel time from the travel times of the route segments.

Returns:
estimated traveling time of the route in seconds

getGeographicArea

```
public GeographicArea getGeographicArea(float direction)
  throws ServiceException
```

Returns the `GeographicArea` of the route. The geographic area is the smallest rectangle that can be drawn around the route. The area may also be circular or polygon, it the route can be better fitted into them. The `direction` given as parameter is only a hint for the API implementation about the direction of the area when forming the geographical area. The direction represents the screen up direction of the terminal. The two sides of the bounding rectangle should be parallel to the given direction. The direction is given as degrees from true north. Valid direction values are `[0, 360)`.

(continued from last page)

Parameters:
`direction` - a direction of the two sides in the formed rectangle

Returns:
a `GeographicArea` objects that represents an area that contains the route

Throws:
`java.lang.IllegalArgumentException` - if `direction` < 0 or `direction` 360
`ServiceException` - if geographic area cannot be created, eg. for a reason that given direction is not supported.

getRouteType

`public String getRouteType()`

Returns the type of the route that is used in the route calculations by the navigation service provider. The possible return values depend on the navigation service provider. The basic route types that can be used as return values are defined as constants in `RoutePreferences` class and they start with `ROUTE_TYPE_` prefix.

Returns:
the type of the route, `null` if the navigation service provider does not support route type selection

setRouteType

`public void setRouteType(String routeType)`

Sets the type of the route. The basic route types that can be used as return values are defined as constants in `RoutePreferences` class and they start with `ROUTE_TYPE_` prefix.

Parameters:
`routeType` - the type of the route

getStartingPoint

`public Coordinates getStartingPoint()`

Returns the coordinates of the starting point of the route. The starting point is the first coordinate in the first route segment.

Returns:
the starting point coordinates

getDestinationPoint

`public Coordinates getDestinationPoint()`

Returns the coordinates of the destination point of the route. The destination point is the last coordinate in the last route segment.

Returns:
the destination point coordinates

com.sirf.microedition.location.services
Class RoutePreferences

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.RoutePreferences
```

public class **RoutePreferences**
extends Object

This class capsulates the preferences that are used when requesting routing services from a navigation service provider. The preferences include, for example, the type of the route, the transport mode used to travel the route and the navigation type used when calculating the route.

This class defines a set of basic transport types, route and navigation types. Because the list of transport modes, route and navigation types given in this class are not complete, the API implementation and navigation service provider may extend these lists. The API implementation and a navigation service provider **may** also add new preferences. This class provides a mechanism to define additional preference properties. Property keys for the additional route preferences defined by each navigation service provider can be retrieved with `ProviderInfo.getPropertyKeys` method.

An application can query the supported transport types, route and navigation types with the `ProviderInfo.getProperty(String key)` method. The keys for these properties are defined as constants in `ProviderInfo` class.

Field Summary

`public static final`	`NAVIGATION_TYPE_REAL_TIME` Navigation service type that gives real-time navigation information. Value: **`real_time`**
`public static final`	`NAVIGATION_TYPE_TURN_BY_TURN` Navigation service type that requests turn-by-turn instructions to navigate through the route Value: **`turn_by_turn`**
`public static final`	`NAVIGATION_TYPE_VOICE` Navigation service type that gives voice instructions to navigate through the route Value: **`voice`**
`public static final`	`ROUTE_TYPE_FASTEST` Route type for the fastest route Value: **`fastest`**
`public static final`	`ROUTE_TYPE_LEAST_TRAFFIC` Route type for a low traffic route Value: **`least_traffic`**
`public static final`	`ROUTE_TYPE_SCENIC` Route type for a scenic route Value: **`most scenic`**
`public static final`	`ROUTE_TYPE_SHORTEST` Route type for shortest route Value: **`shortest`**

public static final	TRANSPORT_BICYCLE A constant for a bicycle transport mode Value: **transport_bicycle**
public static final	TRANSPORT_CAR A constant for car transport mode Value: **transport_car**
public static final	TRANSPORT_PEDESTRIAN A constant for a pedestrian transport mode Value: **transport_pedestrian**
public static final	TRANSPORT_PUBLIC A constant for a public transport mode Value: **transport_public**

Constructor Summary

public	RoutePreferences() Constructs an empty RoutePreferences object.
public	RoutePreferences(String routeType, String transportMode, String[] navigationType, String[] avoidFeatures, GeographicArea[] avoidAreas) Constructs a RoutePreferences object.

Method Summary

GeographicArea[]	getAvoidAreas() Returns an array of geographical areas that should be avoided in the navigation and route generation.
String[]	getAvoidFeatures() Returns an array of features that should be avoided in the navigation and route generation.
String[]	getNavigationType() Returns an array of preferred navigation types set by the application for the navigation and route generation.
Object	getProperty(String key) Returns the value of the requested property as an object.
String	getRouteType() Returns the preferred route type set by the application for the navigation and route generation.
String	getTransportMode() Returns the preferred transport mode set by the application for the navigation and route generation.
void	setAvoidAreas(GeographicArea[] areas) Sets the avoid areas to be used in navigation and route generation.
void	setAvoidFeatures(String[] features) Sets the avoid features to be used in navigation and in route generation.
void	setNavigationType(String[] navigationType) Sets the navigation types to be used in navigation and route generation.

void	setProperty(String key, Object value) Sets the property for specified key.
void	setRouteType(String routeType) Sets the route type to be used in navigation and route generation.
void	setTransportMode(String transportMode) Sets the transport mode to be used in navigation and route generation.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

TRANSPORT_CAR

`public static final java.lang.String` **`TRANSPORT_CAR`**

A constant for car transport mode
Constant value: **`transport_car`**

TRANSPORT_BICYCLE

`public static final java.lang.String` **`TRANSPORT_BICYCLE`**

A constant for a bicycle transport mode
Constant value: **`transport_bicycle`**

TRANSPORT_PUBLIC

`public static final java.lang.String` **`TRANSPORT_PUBLIC`**

A constant for a public transport mode
Constant value: **`transport_public`**

TRANSPORT_PEDESTRIAN

`public static final java.lang.String` **`TRANSPORT_PEDESTRIAN`**

A constant for a pedestrian transport mode
Constant value: **`transport_pedestrian`**

ROUTE_TYPE_FASTEST

`public static final java.lang.String` **`ROUTE_TYPE_FASTEST`**

Route type for the fastest route
Constant value: **`fastest`**

ROUTE_TYPE_SHORTEST

`public static final java.lang.String` **`ROUTE_TYPE_SHORTEST`**

Route type for shortest route
Constant value: **`shortest`**

ROUTE_TYPE_SCENIC

```
public static final java.lang.String ROUTE_TYPE_SCENIC
```

Route type for a scenic route
Constant value: **`most scenic`**

ROUTE_TYPE_LEAST_TRAFFIC

```
public static final java.lang.String ROUTE_TYPE_LEAST_TRAFFIC
```

Route type for a low traffic route
Constant value: **`least_traffic`**

NAVIGATION_TYPE_VOICE

```
public static final java.lang.String NAVIGATION_TYPE_VOICE
```

Navigation service type that gives voice instructions to navigate through the route
Constant value: **`voice`**

NAVIGATION_TYPE_TURN_BY_TURN

```
public static final java.lang.String NAVIGATION_TYPE_TURN_BY_TURN
```

Navigation service type that requests turn-by-turn instructions to navigate through the route
Constant value: **`turn_by_turn`**

NAVIGATION_TYPE_REAL_TIME

```
public static final java.lang.String NAVIGATION_TYPE_REAL_TIME
```

Navigation service type that gives real-time navigation information. This navigation type is able to receive information about the current location and act based on it.
Constant value: **`real_time`**

Constructors

RoutePreferences

```
public RoutePreferences()
```

Constructs an empty `RoutePreferences` object. The values for different preferences can later be assigned with `set*` methods in this class.

RoutePreferences

```
public RoutePreferences(String routeType,
                        String transportMode,
                        String[] navigationType,
                        String[] avoidFeatures,
                        GeographicArea[] avoidAreas)
```

Constructs a `RoutePreferences` object. Values for different preferences are given as parameters. Possible values for each of the parameters are properties of the navigation service provider. They can be retrieved with `ProviderInfo.getProperty(key)` method where the key values are `ProviderInfo.NAV_SP_SUPPORTED_ROUTE_TYPES`, `ProviderInfo.NAV_SP_SUPPORTED_TRANSPORT_MODES`, `ProviderInfo.NAV_SP_SUPPORTED_NAVIGATION_TYPES` `ProviderInfo.NAV_SP_SUPPORTED_AVOID_FEATURES` for the corresponding input parameters. Passing `null` as any of the parameters means that there is no preference for that setting.

(continued from last page)

Parameters:

`routeType` - the type of the route
`transportMode` - transport mode preferred for the navigation
`navigationType` - an array of the preferred navigation types
`avoidFeatures` - an array of features to be avoided in navigation and route generation
`avoidAreas` - geographical areas to be avoided in navigation and route generation

Methods

setTransportMode

`public void **setTransportMode**(String transportMode)`

Sets the transport mode to be used in navigation and route generation. Possible transport mode values can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_TRANSPORT_MODES`. Passing `null` as the parameter means that there is no preference for transport mode.

Parameters:

`transportMode` - transport mode to be used in navigation and route generation

---

getTransportMode

`public String **getTransportMode**()`

Returns the preferred transport mode set by the application for the navigation and route generation. The returned value has been set either in the constructor or with `setTransportMode` method. The returned value is one of the possible transport mode values that can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_TRANSPORT_MODES` or `null` if no preference has been set.

Returns:

the preferred transport mode, `null` if no preference for transport mode has been set

---

setRouteType

`public void **setRouteType**(String routeType)`

Sets the route type to be used in navigation and route generation. Possible route type values can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_ROUTE_TYPES`. Passing `null` as the parameter means that there is no preference for route type.

Parameters:

`routeType` - the type of the requested route // * @throws java.lang.IllegalArgumentException if `routeType` // * is not one of the values retrieved with // * `ProviderInfo.getProperty(key)` method

---

getRouteType

`public String **getRouteType**()`

Returns the preferred route type set by the application for the navigation and route generation. The returned value has been set either in the constructor or with `setRouteType` method. The returned value is one of the possible route type values that can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_ROUTE_TYPES` or `null` if no preference has been set.

Returns:

the preferred route type, `null` if no preference has been set

(continued from last page)

setNavigationType

`public void setNavigationType(String[] navigationType)`

Sets the navigation types to be used in navigation and route generation. If an application wants to get more than one kind on navigation, it **may** include several navigation types into the input array. The array is in the preference order, so the most preferred navigation type is in the first element of the array. Possible navigation type values can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_NAVIGATION_TYPES`. Passing `null` as the parameter means that there is no preference for navigation type.

Parameters:

`navigationType` - an array of the preferred navigation types

Throws:

`java.lang.IllegalArgumentException` - if items in the `navigationType` array contains a navigation type that is not one of the values retrieved with `ProviderInfo.getProperty(key)`

getNavigationType

`public String[] getNavigationType()`

Returns an array of preferred navigation types set by the application for the navigation and route generation. The returned array has been set either in the constructor or with `setNavigationType` method. The possible navigation type values in the returned array can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_NAVIGATION_TYPES` or `null` if no preference has been set.

Returns:

an array of preferred navigation types, `null` if no preference for navigation type has been set

setAvoidFeatures

`public void setAvoidFeatures(String[] features)`

Sets the avoid features to be used in navigation and in route generation. These avoid features can be, for example, toll ways or bridges. These features are navigation provider specific and can be retrieved with `ProviderInfo.getProperty()` method using key `NAV_SP_SUPPORTED_AVOID_FEATURES`. Passing `null` as the parameter means that there are no features to be avoided. If `features` is empty, that is interpreted as passing `null` as the parameter.

Parameters:

`features` - the features to be avoided in navigation and route generation

Throws:

`java.lang.IllegalArgumentException` - if items in the `features` array contains a feature that is not one of the values retrieved with `ProviderInfo.getProperty(key)`

getAvoidFeatures

`public String[] getAvoidFeatures()`

Returns an array of features that should be avoided in the navigation and route generation. The returned array has been set either in the constructor or with `setAvoidFeatures` method. The possible avoid features in the returned array can be retrieved with `ProviderInfo.getProperty(key)` method where the key value is `ProviderInfo.NAV_SP_SUPPORTED_AVOID_FEATURES` or `null` if no preference has been set.

Returns:

an array of the avoid features, `null` if no avoid features for navigation and route generation have been set

setAvoidAreas

`public void setAvoidAreas(GeographicArea[] areas)`

Sets the avoid areas to be used in navigation and route generation. Passing `null` as the parameter means that there are no areas to be avoided. If `areas` is empty, that is interpreted as passing `null` as the parameter.

Parameters:

`areas` - geographical areas to be avoided in navigation and route generation

getAvoidAreas

`public GeographicArea[] getAvoidAreas()`

Returns an array of geographical areas that should be avoided in the navigation and route generation. The returned array has been set either in the constructor or with `setAvoidAreas` method. If no preference has been set, `null` is returned.

Returns:

an array of the geographical area to be avoided, `null` if no avoided areas have been set

getProperty

`public Object getProperty(String key)`

Returns the value of the requested property as an `object`. Returned value is the value set by the API implementation.

Parameters:

`key` - the identifier of the property

Returns:

value of the property, `null` if the `key` is not defined

setProperty

```
public void setProperty(String key,
          Object value)
```

Sets the property for specified key.

Parameters:

`key` - the key identifier of the property

`value` - the value of the property

Throws:

`java.lang.IllegalArgumentException` - if `key` is not one of the values retrieved with `getPropertyKeys` method

`java.lang.NullPointerException` - if value is `null`

com.sirf.microedition.location.services
Class RouteSegment

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.RouteSegment
```

public class **RouteSegment**
extends Object

This class represents a segment of a route. It has a set of coordinates that form the geometry of the segment. The segment also contains a description, a length and an estimated time for traveling it. There is also a possibility to include instructions for navigating on the route segment as well as the locations where those instructions should be given.

Constructor Summary	
`public`	`RouteSegment(Coordinates[] coords)` Constructor to create a route segment.

Method Summary	
`void`	`addCoordinates(Coordinates coordinates)` Adds the specified coordinates to the end of the `Coordinates` array of the route segment.
`void`	`addCoordinates(Coordinates coordinates, int index)` Adds the specified coordinates to the specified indes in the `Coordinates` array of the route segment.
`void`	`addInstruction(String instruction, Coordinates location)` Adds a new navigation instruction to the route segment.
`void`	`addInstruction(String instruction, Coordinates location, int index)` Adds the specified instruction and location to the specified indes in the navigation instructions of the route segment.
`String`	`getDescription()` Returns the description given for the route segment.
`GeographicArea`	`getGeographicArea(float direction)` Returns the `GeographicArea` of the route segment.
`Coordinates[]`	`getGeometry()` Returns the geometry of the route segment.
`Coordinates[]`	`getInstructionLocations()` Returns an array of `Coordinate` objects that specify the locations where the instructions for navigating the route segment should be given.
`String[]`	`getInstructions()` Returns an array of `String` objects that contain the instructions to navigate through a route segment.
`double`	`getLength()` Returns the length of the route segment in meters.

String	`getTransportMode()` Returns the transport mode that the navigation service provider defines as the best transport mode for this route segment.
long	`getTravelTime()` Returns the estimated time it takes to travel the route segment.
void	`setDescription(String description)` Sets the description given for the route segment.
void	`setLength(double length)` Sets the length of this RouteSegment.
void	`setTransportMode(String mode)` Sets the transport mode for this route segment.
void	`setTravelTime(long travelTime)` Sets the estimated time it takes to travel the route segment.

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

Constructors

RouteSegment

`public RouteSegment(Coordinates[] coords)`

Constructor to create a route segment. The segment contains all the coordinates given in the `coordinates` parameter. The `coordinates` array **must** have at least two points, the start and end point of the route segment.

Parameters:

`coordinates` - the coordinates of the route segment

Throws:

`java.lang.NullPointerException` - if `coordinates` is `null`
`java.lang.IllegalArgumentException` - if length of the `coordinates` array is < 2

Methods

getDescription

`public String getDescription()`

Returns the description given for the route segment. This description may contain some characteristics about the segment or other additional information about the route segment. This information may be shown to the user.

Returns:

the description of the segment, null if description is not set.

setDescription

`public void setDescription(String description)`

(continued from last page)

Sets the description given for the route segment. This description may contain some characteristics about the segment or other additional information about the route segment. This information may be shown to the user.

Parameters:
`description` - the description of the segment

getGeometry

`public Coordinates[] getGeometry()`

Returns the geometry of the route segment. The geometry is the set of coordinates that form the route segment. The implementation **must** guarantee that the returned array contains enough `Coordinates` objects so that the route geometry can be presented by connecting the coordinates with straight lines. The first item in the array **must** be the starting point of the segment.

Returns:
an array of `Coordinates` objects that form the geometry of the segment

getLength

`public double getLength()`

Returns the length of the route segment in meters.

Returns:
the length of the segment in meters

setLength

```
public void setLength(double length)
  throws IllegalArgumentException
```

Sets the length of this RouteSegment.

Parameters:
`length` - Segment length to be set.

Throws:
`IllegalArgumentException` - if length < 0

getTravelTime

`public long getTravelTime()`

Returns the estimated time it takes to travel the route segment. The time is given in seconds.

Returns:
the estimated time of the segment in seconds, -1 if travel time is not available

setTravelTime

`public void setTravelTime(long travelTime)`

Sets the estimated time it takes to travel the route segment. The time is given in seconds.

Parameters:
`travelTime` - the estimated time of the segment in seconds

Throws:

(continued from last page)

`java.lang.IllegalArgumentException` - if `travelTime` < 1

getGeographicArea

```
public GeographicArea getGeographicArea(float direction)
  throws ServiceException
```

Returns the `GeographicArea` of the route segment. The geographic area is the smallest rectangle that can be drawn around the route segment. The area may also be circular or polygon, it the route segment can be better fitted into them. The `direction` given as parameter is only a hint for the API implementation about the direction of the area when forming the geographical area. The direction represents the screen up direction of the terminal. if returned area is a rectangle, the two sides of the bounding rectangle should be parallel to the given direction. The direction is given as degrees from true north. Valid direction values are `[0, 360)`.

Parameters:

`direction` - a direction of the two sides in the formed rectangle

Returns:

a `GeographicArea` objects that represents an area that contains the route

Throws:

`java.lang.IllegalArgumentException` - if `direction` < 0 or `direction` 360

`ServiceException` - if geographic area cannot be created, eg. for a reason that given direction is not supported.

addCoordinates

```
public void addCoordinates(Coordinates coordinates)
```

Adds the specified coordinates to the end of the `Coordinates` array of the route segment. If `coordinates` parameter is `null`, this method returns without actions.

Parameters:

`coordinates` - coordinates to be added to the route segment

addCoordinates

```
public void addCoordinates(Coordinates coordinates,
         int index)
```

Adds the specified coordinates to the specified indes in the `Coordinates` array of the route segment. The `index` **must** be a value greater than or equal to 0 and less than the current number of coordinates in the array. Each coordinates in the array with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously. If `coordinates` parameter is `null`, this method returns without actions.

Parameters:

`coordinates` - coordinates to be added to the route segment

Throws:

`java.lang.IndexOutOfBoundsException` - if `index` is invalid

getTransportMode

```
public String getTransportMode()
```

Returns the transport mode that the navigation service provider defines as the best transport mode for this route segment. Possible values are defined as constants in `RoutePreferences` class and they start with `TRANSPORT_` prefix.

Returns:

the transport mode for this RouteSegmentreturns, null if transport mode is not set.

setTransportMode

```
public void setTransportMode(String mode)
```

Sets the transport mode for this route segment. Possible values are defined as constants in `RoutePreferences` class and they start with `TRANSPORT_` prefix.

Parameters:

`mode` - the transport that used in this route segment

Throws:

`java.lang.IllegalArgumentException` - if `mode` is not one of the transport mode constants defined in `RoutePreferences` class

getInstructionLocations

```
public Coordinates[] getInstructionLocations()
```

Returns an array of `Coordinate` objects that specify the locations where the instructions for navigating the route segment should be given.

Returns:

locations where to give instructions for the route segment, an empty array, if no locations for the instructions are provided

getInstructions

```
public String[] getInstructions()
```

Returns an array of `String` objects that contain the instructions to navigate through a route segment.

Returns:

an array of instructions for a route segment

addInstruction

```
public void addInstruction(String instruction,
        Coordinates location)
```

Adds a new navigation instruction to the route segment. The instuction and the coordinates where it should be given are passed in as parameters.

Parameters:

`instruction` - the instuction for navigation

`location` - the location where the instruction should be given

Throws:

`java.lang.NullPointerException` - if `instruction` is `null`

addInstruction

```
public void addInstruction(String instruction,
        Coordinates location,
        int index)
```

Adds the specified instruction and location to the specified indes in the navigation instructions of the route segment. The `index` **must** be a value greater than or equal to 0 and less than the current number of instructions in the array. Each instruction in the array with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

(continued from last page)

Parameters:

`instruction` - instruction to be added to the route segment
`location` - location for instructions to be added to the route segment

Throws:

`java.lang.IndexOutOfBoundsException` - if `index` is invalid
`java.lang.NullPointerException` - if `instruction` is `null`

com.sirf.microedition.location.services
Class ServiceException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-com.sirf.microedition.location.services.ServiceException
```

All Implemented Interfaces:
Serializable

public class **ServiceException**
extends Exception

The `ServiceException` is thrown when an error related to handling service provider requests has occurred. This error situation can be, for example, that the service provider does not support requested service or it does not have the needed maps or navigation information to serve the request.

Constructor Summary

`public`	`ServiceException()` Constructs a `ServiceException` with no detail message.
`public`	`ServiceException(String s)` Constructs a `ServiceException` with the specified detail message.

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, toString, wait, wait, wait`

Constructors

ServiceException

`public ServiceException()`

Constructs a `ServiceException` with no detail message.

ServiceException

`public ServiceException(String s)`

Constructs a `ServiceException` with the specified detail message.

Parameters:

(continued from last page)

s - the detailed message

com.sirf.microedition.location.services
Interface ServiceListener

All Subinterfaces:
NavigationServiceListener, MapServiceListener

---

public interface **ServiceListener**
extends

This is a super interface for all listeners related to different services. It contains the common methods for all service listeners.

When a callback method in this interface is called, the service request has ended for some reason. This means that the application then gets back the focus on the user interface and should behaive accoringly.

Method Summary

`void`	`requestCanceled(ServiceProvider provider)` Called by the platform when the service request has been cancelled by the user.
`void`	`requestCompleted(ServiceProvider provider)` Called by the platform when the service requested by the application has been successfully completed.
`void`	`requestError(ServiceProvider provider)` Called by the platform when there is an unexpected error situations, like loss of network connection, in the service provider that prevents it from finishing the service request.
`void`	`requestReset(ServiceProvider provider)` Called by the platform when the service request has been cancelled by the application.
`void`	`requestTimeout(ServiceProvider provider)` Called by the platform when the timeout set for a service request has been reached and the request has not been finished.

Methods

requestCompleted

`public void requestCompleted(ServiceProvider provider)`

Called by the platform when the service requested by the application has been successfully completed.

Parameters:
`provider` - the provider whose request has been completed

---

requestCanceled

`public void requestCanceled(ServiceProvider provider)`

Called by the platform when the service request has been cancelled by the user. This can happen for example, when a service provider is handling asynchronous service request and showing sime dialog to the user and the user cancels the whole operation from the dialog.

(continued from last page)

Parameters:

`provider` - the provider whose request has been canceled

requestReset

`public void **requestReset**(ServiceProvider provider)`

Called by the platform when the service request has been cancelled by the application.

Parameters:

`provider` - the provider whose request has been reset

requestTimeout

`public void **requestTimeout**(ServiceProvider provider)`

Called by the platform when the timeout set for a service request has been reached and the request has not been finished.

Parameters:

`provider` - the provider whose request is taking too long

requestError

`public void **requestError**(ServiceProvider provider)`

Called by the platform when there is an unexpected error situations, like loss of network connection, in the service provider that prevents it from finishing the service request.

Parameters:

`provider` - the provider who had errornous situation

com.sirf.microedition.location.services
Interface ServiceProvider

All Subinterfaces:
NavigationServiceProvider, MapServiceProvider, GeocodingServiceProvider, POIServiceProvider

public interface **ServiceProvider**
extends

This is a super interface for all service providers defined in this API. It contains the methods common to all providers.

There can be situations where there are several applications running and they all need services from the same service provider. When an application requests an instance of a service provider, the API implementation **must** return different instance of the service provider to requesting applications. However at the lower levels, there **may** be only one service implementation that handles all the service requests. In these situations the API implementation **must** decide how to handle possibly concurrent service request from multiple applications. Some implementations may want to queue the requests, where as some other implementation may override the service request or ask the user what to do.

Method Summary

`void`	`configureProvider()` With this method the application can request the service provider to show it's own configuration to the user.
`String`	`getLanguage()` Returns the preferred language to be used in the service requests.
`ProviderInfo`	`getProviderInfo()` Returns information about the service provider.
`int`	`getTimeout()` Returns the timeout that has been set for the maximum time to serve the service request.
`void`	`reset()` Requests the service provider to reset.
`void`	`setLanguage(String language)` Sets the language to be used in service requests.
`void`	`setTimeout(int timeout)` This method is used to set maximum timeout for the service provider to serve service requests.
`boolean`	`supportsArea(GeographicArea area)` With this method an application can check if a service provider has a information about the specified geographical area.

Methods

getProviderInfo

```
public ProviderInfo getProviderInfo()
```

(continued from last page)

Returns information about the service provider. This in information contains for example the type of the service and the name of the service provider.

Returns:
the service provider info

reset

`public void **reset**()`

Requests the service provider to reset. If the service provider has a dialog ongoing, reset will happen once the dialog is finished. All pending service requests will be aborted. This method may be used with both asynchronous and synchronous service requests.

In the asynchronous service requests the application is notified from the reset request through `ServiceListener.requestReset(ServiceProvider)` callback method.

In the synchronous service requests the method throws a `InterrupedException` when this methods is called.

setLanguage

`public void **setLanguage**(String language)`

Sets the language to be used in service requests. Possible language tag values can be retrieved with `ProviderInfo.getProperty(key)` method where the key values is `SUPPORTED_LANGUAGES`.

Parameters:
`language` - the language to be used in navigation

Throws:
`java.lang.IllegalArgumentException` - if `language` is not one of the values retrieved with `ProviderInfo.getProperty(key)` method

getLanguage

`public String **getLanguage**()`

Returns the preferred language to be used in the service requests. The returned value has been set with `setLanguage` method. The returned value is one of the possible language tag values that can be retrieved with `ProviderInfo.getProperty(key)` method where the key values is `SUPPORTED_LANGUAGES` or `null` if no preferred language has been set.

Returns:
the preferred language, `null` if no preference for language has been set

setTimeout

`public void **setTimeout**(int timeout)`

(continued from last page)

This method is used to set maximum timeout for the service provider to serve service requests. These service requests include both asynchronous and synchronous methods. The change in the timeout affects only the service requests that are started after this method is called.

When this timeout is reached and the service request has not been completed, the API implementation **must** reset the service request. If the service provider has a dialog ongoing, reset will happen once the dialog is finished. If the service request is asynchronous an application is notified though `ServiceListener.requestTimeout(ServiceProvider)` notification. In the synchronous requests, the blocking method throws an `InterruptedException` when the timeout is reached.

Parameters:
`timeout` - the maximum execution time for the service request in seconds

Throws:
`java.lang.IllegalArgumentException` - if `timeout 0`

---

getTimeout

`public int getTimeout()`

Returns the timeout that has been set for the maximum time to serve the service request. The API implementation **must** have a default value for the timeout, but it **may** be changed by the application using `setTimeout(timeout)` method.

Returns:
the maximum service request execution time in seconds

---

supportsArea

`public boolean supportsArea(GeographicArea area)`

With this method an application can check if a service provider has a information about the specified geographical area. The information may be geocoding data, maps or navigation information depending on the type of the service provider. With this method the check can be made before connecting to any particular service provider.

The implementation of this method may approximate the given area with a rectangular area around the given geographic area and check whether that approximated area is supported. Therefore in some cases this method may return `true`, but the actual method that require the information for the specified area return `null` or throws an exception. This is true also for circular and polygonal areas.

Parameters:
`area` - geographical area from which the information is needed

Returns:
`true` if the service provider has a information for the requested geographical area, else `false`

Throws:
`java.lang.NullPointerException` - if `area` is `null`

---

configureProvider

`public void configureProvider()`

With this method the application can request the service provider to show it's own configuration to the user. An application can check if the service provider supports this configuration UI with `ProviderInfo.getProperty()` method using `CONFIGURATION_UI` key. This method blocks until the configuration has been completed. If the service provider does not support configuration UI, this method returns without any actions.

com.sirf.microedition.location.services
Class StaticMapLayerWithCircleArea

```
java.lang.Object
  |
  +-com.sirf.microedition.location.services.MapLayer
      |
      +-com.sirf.microedition.location.services.StaticMapLayerWithCircleArea
```

public class **StaticMapLayerWithCircleArea**
extends MapLayer

Method Summary

void	centerOnCoordinates(Coordinates newCoordinates)
int	getCurrentZoomLevel()
GeographicArea	getGeographicArea()
int	getHeight()
int	getWidth()
boolean	isValidZoomlevel(int z)
void	networkRequestCompleted(NetworkMessage req)
void	pan(int x, int y)
void	renderMap(Object graphics, boolean needCompleteImage)
void	rotate(int angle)
void	setGeographicArea(GeographicArea area)
void	setMapListener(MapServiceListener listener)
int[]	transformToImageCoordinates(Coordinates coordinates)
Coordinates	transformToWGS84Coordinates(int x, int y)
void	zoom(int newZoomLevel)

Methods inherited from class com.sirf.microedition.location.services.MapLayer `centerOnCoordinates, getCurrentZoomLevel, getGeographicArea, getHeight, getWidth, isValidZoomlevel, pan, renderMap, rotate, setGeographicArea, setMapListener, transformToImageCoordinates, transformToWGS84Coordinates, zoom`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

renderMap

```
public void renderMap(Object graphics,
         boolean needCompleteImage)
  throws ServiceException
```

Renders this map to the provided graphics context passed in as parameter `graphics`. The API implementation draws a map image in size that was specified when retrieving this `MapLayer` instance. This method blocks until the rending is done.

With parameter `needCompleteImage` an application can control whether the map image rendered to the graphics context is complete or if it can be be updated later when data from the network is available. If an application is able to handle incomplete images, it should register `MapServiceListener` to recieve notifications when new map data is available.

transformToWGS84Coordinates

```
public Coordinates transformToWGS84Coordinates(int x,
         int y)
```

Converts a point on the map image into coordinates in the WGS84 projection. `x` and `y` are presented in the coordinates system of the map image. The origin point is the bottom left corner of the map image. The values on the `x` axis grow horizontally towards the right edge of the map image and on the `y` axis vertically towards to the bottom edge of the map image.

This method does not include any information about the altitude into the returned `Coordinates` object. Therefore the altitude is set to `Float.NaN`.

transformToImageCoordinates

```
public int[] transformToImageCoordinates(Coordinates coordinates)
```

Converts coordinates in WGS84 projection system into the coordinate system of the map image. The first element in the returned array contains the point on the `x` axis and the second element is the point on the `y` axis of the map image. The origin point is the top left corner of the map image. The values on the `x` axis grow to right and on the `y` axis grow to down.

pan

```
public void pan(int x,
         int y)
  throws ServiceException
```

Pans the map layer to the given direction.

(continued from last page)

rotate

```
public void rotate(int angle)
  throws ServiceException
```

Rotates the map layer to the given angle. The angle is given as degrees from true north.

isValidZoomlevel

```
public boolean isValidZoomlevel(int z)
```

Checks if passed zoom level is valid with this map layer.

zoom

```
public void zoom(int newZoomLevel)
  throws ServiceException
```

Zooms the map layer to the given zoom level. The zoom levels supported by the map service provider can be retrieved from the `ProviderInfo.getProperty` with key `MAP_SP_ZOOM_LEVELS`.

getCurrentZoomLevel

```
public int getCurrentZoomLevel()
```

Returns the zoom level that is currently used in this map layer. The return value is on of the constants retrieved with `ProviderInfo.getProperty(key)` method where the key value is `MAP_SP_ZOOM_LEVELS`.

getGeographicArea

```
public GeographicArea getGeographicArea()
```

Returns the `GeographicArea` object that this map object currently represents.

setGeographicArea

```
public void setGeographicArea(GeographicArea area)
  throws ServiceException
```

Set a new geographic area for the map layer. The API implementation may do some scaling or panning so that it will be able to render the layer.

centerOnCoordinates

```
public void centerOnCoordinates(Coordinates newCoordinates)
  throws ServiceException
```

Centers the map layer to the given coordinates. The zoom level is not changed in this operation.

setMapListener

```
public void setMapListener(MapServiceListener listener)
```

Sets a `MapServiceListener` to be called whenever API implementation has received map new data from the network.

getWidth

```
public int getWidth()
```

(continued from last page)

Returns the width of the map layer image as pixels. This is the same value that when requesting this map layer from the service provider with `MapServiceProvider.getMapLayer` method.

getHeight

`public int getHeight()`

Returns the height of the map layer image as pixels. This is the same value that when requesting this map layer from the service provider with `MapServiceProvider.getMapLayer` method.

networkRequestCompleted

`public void networkRequestCompleted(NetworkMessage req)`

Index

A

addCategory 40
addCoordinates 159
addInstruction 160
addLandmark 35
addLandmarkStoreListener 41
addNMEAListener 76
addProximityListener 59
AddressInfo 7
addressInfo 47
addSatlliteInfoListener 86
AVAILABLE 55
azimuthTo 14

B

backgroungImage 91
BTGPSDiscoverer 70
BUILDING_FLOOR 6
BUILDING_NAME 6
BUILDING_ROOM 6
BUILDING_ZONE 7

C

centerOn 98, 106, 127, 144
centerOnCoordinates 110, 171
CircleGeographicArea 96
CITY 6
close 71
COM_SIRF_STUDIO_MAP_LAYERTYPE 133
COM_SIRF_STUDIO_MAP_LAYERTYPE_STATIC 134
COM_SIRF_STUDIO_MAP_LAYERTYPE_TILING 134
COM_SIRF_STUDIO_MAP_PROJECTION 134
COM_SIRF_STUDIO_MAP_PROJECTION_LINEAR 134
COM_SIRF_STUDIO_MAP_PROJECTION_MERCATOR 134
COM_SIRF_STUDIO_MAP_TILED_CONTENT_VERSION 134
COM_SIRF_STUDIO_MAP_TILED_TILESIZE 134
COM_SIRF_STUDIO_MAP_TILED_ZOOMLEVELS 134
COM_SIRF_STUDIO_NAV_MAPPROV_ID 135
COM_SIRF_STUDIO_POI_CATEGORIES 87
COM_SIRF_STUDIO_POI_CATEGORY_ID 88
COM_SIRF_STUDIO_SP_DEFAULT_VIEWPORT 133
COM_SIRF_STUDIO_SP_ID 133
COM_SIRF_STUDIO_SP_NAME 133
COM_SIRF_STUDIO_SP_SUPPORTED_GEOGRAPHIC_AREA 133
COM_SIRF_STUDIO_SP_URL 133
CONFIGURATION_UI 131
configureProvider 168
connectToServiceProvider 88, 89, 139, 140
containsCoordinates 96, 105, 125, 142
convert 12, 14
Coordinates 10
coordinates 47
COUNTRY 6
COUNTRY_CODE 6
COUNTY 6
course 47
CREATE_NETWORK_TRAFFIC 131
createLandmarkStore 34
Criteria 18
CROSSING1 7
CROSSING2 7

D

DD_MM 10
DD_MM_SS 10
deleteCategory 40
deleteLandmark 39
deleteLandmarkStore 35
destinationReached 119
display 91
displayMap 116
distance 14
DISTRICT 6
DOWN 104

E

emulateNetworkFailure 92
equals 29
EXTENSION 5
extraInfo 48

F

findServiceProviders 88, 138

G

GEO_SP_SUPPORTED_GEOCODING 131
GEO_SP_SUPPORTED_REVERSE_GEOCODING 131
geocode 100
GEOCODING 138
getAddressInfo 27, 49
getAltitude 11
getAuthor 27
getAvoidAreas 155
getAvoidFeatures 154
getAzimuth 83
getBoundingBox 98, 106, 127, 144
getCategories 40, 79
getCategoriesOfLandmark 41
getCenterPointCoords 98, 127, 144
getCompassAzimuth 62
getCoordinates 125, 144
getCourse 49
getCurrentZoomLevel 110, 171
getDescription 26, 157
getDestinationPoint 148
getDirection 96, 105, 126, 142
getElevationAngle 83
getExtraInfo 27, 49
getField 7
getGeographicArea 108, 147, 159, 171
getGeometry 158
getHeight 110, 172
getHorizontalAccuracy 19, 67
getIdentifier 27
getInstance 34, 56, 85
getInstructionLocations 160
getInstructions 160
getLandmark 37
getLandmarks 36, 37
getLanguage 167
getLastKnownLocation 58
getLatitude 11
getLength 146, 158
getLocation 56
getLocationMethod 49
getLongitude 11
getMapLayer 117
getName 26, 135
getNavigationType 154
getNumberOfLandmarks 38
getNumSatsUsed 80, 85
getOrientation 63
getPitch 63
getPOIs 78
getPreferredPowerConsumption 19
getPreferredResponseTime 19
getProperty 135, 155
getPropertyKeys 136
getProviderInfo 140, 166
getQualifiedCoordinates 27, 48
getRadius 98
getRoll 63
getRoute 122
getRouteType 148, 153
getSatelliteId 82
getSatelliteInfo 81
getSegments 146
getSignalStrength 83
getSpeed 49
getStartingPoint 148
getState 55
getSummary 147
getTimeout 168
getTimestamp 27, 48
getTransportMode 153, 159
getTravelTime 147, 158
getType 135
getVerticalAccuracy 19, 67
getWidth 110, 171

H

highlightedLandmarkIcon 92

I

isAddressInfoRequired 20
isAllowedToCost 19

isAltitudeRequired 20
isDebugMode 91
isOrientationMagnetic 62
isSpeedAndCourseRequired 20
isValid 47, 48
isValidZoomlevel 111, 171

L

Landmark 25, 26
LandmarkException 30
landmarkIcon 92
landmarkIconsAnchor 92
landmarkStoreUpdated 42
LEFT 104
listLandmarkStores 35
Location 48
LocationException 51
locationIcon 91
locationMethod 47
LocationProvider 55
locationUpdated 52

M

MAP 138
MAP_SP_RETURN_MAP_OBJECTS 132
MAP_SP_SUPPORTED_MAP_TYPES 132
MAP_SP_ZOOM_LEVELS 132
MAP_TYPE_HYBRID 116
MAP_TYPE_REGULAR 115
MAP_TYPE_SATELLITE 116
mapChanged 112
mapImage 146
mediumFont 91
monitoringStateChanged 64
MTA_ASSISTED 46
MTA_UNASSISTED 47
MTE_ANGLEOFARRIVAL 46
MTE_CELLID 46
MTE_SATELLITE 45
MTE_SHORTRANGE 46
MTE_TIMEDIFFERENCE 45
MTE_TIMEOFARRIVAL 45
MTY_NETWORKBASED 46
MTY_TERMINALBASED 46

N

NAV_SP_GENERATE_ROUTES 132
NAV_SP_SUPPORTED_AVOID_FEATURES 133
NAV_SP_SUPPORTED_NAVIGATION_TYPES 132
NAV_SP_SUPPORTED_ROUTE_TYPES 132
NAV_SP_SUPPORTED_TRANSPORT_MODES 132
navigate 121, 122
NAVIGATION 138
NAVIGATION_TYPE_REAL_TIME 152
NAVIGATION_TYPE_TURN_BY_TURN 152
NAVIGATION_TYPE_VOICE 152
networkRequestCompleted 172
NO_REQUIREMENT 18
notifyDevicesDiscovered 72
notifyDiscoveryError 72

O

Orientation 62
OUT_OF_SERVICE 55

P

pan 97, 104, 109, 126, 143, 170
PHONE_NUMBER 7
POI 88
PolygonGeographicArea 125
POSTAL_CODE 5
POWER_USAGE_HIGH 18
POWER_USAGE_LOW 18
POWER_USAGE_MEDIUM 18
ProviderInfo 135
providerStateChanged 52
proximityEvent 64

Q

QualifiedCoordinates 66

R

RectangleGeographicArea 142

removeLandmarkFromAllCategories 38
removeLandmarkFromCategory 38
removeLandmarkStoreListener 41
removeNMEAListener 76
removeProximityListener 59
removeSatlliteInfoListener 86
renderMap 108, 170
requestCanceled 164
requestCompleted 164
requestError 165
requestReset 165
requestTimeout 165
rescale 97, 104, 126, 143
reset 58, 167
retrieveProviderInfos 140
reverseGeocode 101
RIGHT 104
rotate 97, 105, 109, 126, 143, 170
Route 146
ROUTE_TYPE_FASTEST 151
ROUTE_TYPE_LEAST_TRAFFIC 152
ROUTE_TYPE_SCENIC 152
ROUTE_TYPE_SHORTEST 151
routeCalculated 119
routeChanged 120
routeEndIcon 92
routeIconsAnchor 92
RoutePreferences 152
RouteSegment 157
routeStartIcon 92
routeWaypointIcon 92

S

SatelliteContext 80
SatelliteInfo 82
searchBTGPSDevices 70
SELECT_CIRCLE 115
SELECT_LANDMARKS 115
SELECT_LOCATION 115
SELECT_POLYGON 115
SELECT_RECTANGLE 115
selectedLandmarkIcon 92
selectionUpdated 112
SERVICE_FEE_ATTACHED 131
ServiceException 162
setAddressInfo 28
setAddressInfoRequired 22
setAgpsServer 73
setAltitude 12
setAltitudeRequired 22
setApproxPosition 74
setAuthor 29
setAvoidAreas 155
setAvoidFeatures 154
setCostAllowed 21
setDescription 28, 157
setExtraInfo 29
setField 8
setGeographicArea 109, 171
setHorizontalAccuracy 20, 67
setIdentifier 29
setLanguage 167
setLatitude 12
setLength 158
setLocationListener 57
setLongitude 12
setMapListener 110, 171
setName 28
setNavigationType 153
setPreferredPowerConsumption 21
setPreferredResponseTime 21
setProperty 155
setQualifiedCoordinates 28
setRouteType 148, 153
setSegments 147
setSerialConnectionString 73
setSpeedAndCourseRequired 22
setSummary 147
setTimeout 167
setTransportMode 153, 160
setTravelTime 158
setVerticalAccuracy 21, 68
SiRFProviderManager 88
smallFont 91
speed 47
STATE 6
STREET 5
SUPPORTED_LANGUAGES 131
supportsArea 136, 168

SystemConfiguration 92

T

TEMPORARILY_UNAVAILABLE 55
timestamp 47
toString 8, 15, 29, 98, 144
transformToImageCoordinates 111, 170
transformToWGS84Coordinates 110, 170
TRANSPORT_BICYCLE 151
TRANSPORT_CAR 151
TRANSPORT_PEDESTRIAN 151
TRANSPORT_PUBLIC 151

U

UP 104
updateLandmark 39
updateNMEA 75
updateSatelliteInfo 84
URL 7

W

waypointReached 119

Z

zoom 109, 171