Location Acquisition API
LocationRequestor Python extension for Symbian S60 2nd edition FP2 and higher.

Contents:
1 Terms .......................................................................................................................... 1
2 Constructors/static methods ........................................................................................ 1
   2.1 locationrequestor::LocationRequestor() ............................................................. 1
   2.2 locationrequestor::ShowSatelliteDialog(string nameOfRule) ............................ 1
3 LocationRequestor class ............................................................................................. 2
   3.1 LocationRequestor::SetRequestor(string name) ................................................. 2
   3.2 LocationRequestor::GetNumModules() ............................................................. 2
   3.3 LocationRequestor::GetDefaultModuleId().......................................................... 2
   3.4 LocationRequestor::GetModuleInfoByIndex(int index) .................................... 2
   3.5 LocationRequestor::GetModuleInfoById(int id)................................................ 2
   3.6 LocationRequestor::GetModuleStatus(int id)..................................................... 2
   3.7 LocationRequestor::Open(int moduleId)............................................................ 2
   3.8 LocationRequestor::NotifyPositionUpdate() ...................................................... 3
   3.9 LocationRequestor::GetLastKnownPosition().................................................... 3
   3.10 LocationRequestor::InstallPositionCallback(callback)....................................... 3
   3.11 LocationRequestor::GetSatelliteData(int index) ................................................. 3
   3.12 LocationRequestor::Close() ................................................................................ 4
   3.13 LocationRequestor::SetUpdateOptions(int updateInterval, int updateTimeout,
      int maxAge, int allowPartial)................................................................................. 4
4 Example code.............................................................................................................. 4

1 Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Position module that is used to acquire location information. Examples may be “Internal GPS”, “Bluetooth GPS”, “Network”, …</td>
</tr>
<tr>
<td>Uid</td>
<td>Globally unique identifier</td>
</tr>
<tr>
<td>Long long</td>
<td>A 64 bit long value (also termed int64)</td>
</tr>
</tbody>
</table>

2 Constructors/static methods

2.1 locationrequestor::LocationRequestor()
Constructor, creates a LocationRequestor instance. No arguments.

2.2 locationrequestor::ShowSatelliteDialog(string nameOfRule)
Shows a dialog containing satellite info: which satellites are in view and what are their signal strengths. NameOfRule is requestor data for Location FW. (No, I don’t know what that means either). Only available on 3rd edition and higher.
3 LocationRequestor class

3.1 LocationRequestor::SetRequestor(string name)
Sets the application name that is sent to the underlying location API. This name may be used to attach access rules by the user. The default is “Python Extension”. This method needs to be called before Open, otherwise the name is stored but not used until Close() and then Open() is called again.

3.2 LocationRequestor::GetNumModules()
Gets the number of available position modules.

3.3 LocationRequestor::GetDefaultModuleId()
Gets the uid of the default module.

3.4 LocationRequestor::GetModuleInfoByIndex(int index)
Gets the module info for the module at the specified index, where 0 <= index < GetNumModules(). The result will be a tuple consisting of (in order):

<table>
<thead>
<tr>
<th>Int</th>
<th>Unique identifier of the module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicode string</td>
<td>Name of the module (eg “Internal GPS”)</td>
</tr>
<tr>
<td>Int</td>
<td>Technology type, one or more of the ETechnology* flags</td>
</tr>
<tr>
<td>Int</td>
<td>Device location, one of the EDevice* flags</td>
</tr>
<tr>
<td>Int</td>
<td>Capabilities, one or more of the ECapability* flags</td>
</tr>
<tr>
<td>Long long</td>
<td>Estimated time to first fix, in seconds</td>
</tr>
<tr>
<td>Long long</td>
<td>Estimated time between fixes, in seconds</td>
</tr>
<tr>
<td>Int</td>
<td>Typical horizontal accuracy, in metres</td>
</tr>
<tr>
<td>Int</td>
<td>Typical vertical accuracy, in metres</td>
</tr>
<tr>
<td>Int</td>
<td>Cost indicator, one or more of the ECost* flags</td>
</tr>
<tr>
<td>Int</td>
<td>Power consumption indication, one of the EPower* flags</td>
</tr>
</tbody>
</table>

3.5 LocationRequestor::GetModuleInfoById(int id)
Gets the module info for the module with the specified uid. The result will be a tuple consisting of the same elements as with GetModuleInfoByIndex.

3.6 LocationRequestor::GetModuleStatus(int id)
Gets the status for the module with the specified uid. The result will be a tuple consisting of (in order):

| Int  | Device status, one of the EDevice* flags |
| Int  | Data quality, one of the EDataQuality* flags |

3.7 LocationRequestor::Open(int moduleid)
Opens a connection to the specified module id. Needs to be called before calling NotifyPositionUpdate or GetLastKnownPosition. If the module id is -1 then the default module will be used.
3.8 LocationRequestor::NotifyPositionUpdate()

Will wait for a location update according to the specifications in SetUpdateOptions(). This should be called only when #Open() > #Close(). This will return either an error tuple (in order):

<table>
<thead>
<tr>
<th>Int</th>
<th>Symbian error code (0 = no associated symbian error code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicode string</td>
<td>Error message</td>
</tr>
</tbody>
</table>

or a basic location tuple (in order):

<table>
<thead>
<tr>
<th>Int</th>
<th>Flag if the update contains full information (1) or partial information (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Latitude</td>
</tr>
<tr>
<td>Double</td>
<td>Longitude</td>
</tr>
<tr>
<td>Double</td>
<td>Altitude (metres)</td>
</tr>
<tr>
<td>Double</td>
<td>Horizontal accuracy (metres)</td>
</tr>
<tr>
<td>Double</td>
<td>Vertical accuracy (metres)</td>
</tr>
<tr>
<td>Unicode string</td>
<td>Module name</td>
</tr>
<tr>
<td>Long long</td>
<td>Time of the position (ms since Jan 1st, 1970)</td>
</tr>
</tbody>
</table>

or an extended location tuple that contains the elements of the basic location and in addition (in order, after the basic elements):

<table>
<thead>
<tr>
<th>Double</th>
<th>Speed (km/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Speed accuracy (km/h)</td>
</tr>
<tr>
<td>Double</td>
<td>Heading (degrees)</td>
</tr>
<tr>
<td>Double</td>
<td>Heading accuracy (degrees)</td>
</tr>
<tr>
<td>Double</td>
<td>Satellite time (ms since Jan 1st, 1970)</td>
</tr>
<tr>
<td>Int</td>
<td>Number of satellites in view</td>
</tr>
<tr>
<td>Int</td>
<td>Number of satellites used in the fix (0 = no fix)</td>
</tr>
</tbody>
</table>

3.9 LocationRequestor::GetLastKnownPosition()

Gets the last known position (if any). The result is the same as for NotifyPositionUpdate.

3.10 LocationRequestor::InstallPositionCallback(callback)

Installs a position callback. The callback must be a function that accepts a single argument. An asynchronous location request is scheduled with the location acquisition API and the callback will be called once data becomes available. The callback argument will contain the same data as described in NotifyPositionUpdate.

Note:
- SetUpdateOptions determines the notification frequency etc.
- You need to make sure (through Ao_wait or an Ao_lock) that the asynchronous request will have enough ‘CPU time’ to process.

3.11 LocationRequestor::GetSatelliteData(int index)

Gets the satellite data (position in the sky, signal strength, satellite id) for the last requested location (through GetLastKnownPosition, NotifyPositionUpdate or from within the callback method set using InstallPositionCallback.)
Note that this data is only available if the location module used is satellite based (as GPS is). If the operation is successful, the result will be a tuple consisting of (in order):

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>Satellite ID (in GPS, the PRN).</td>
</tr>
<tr>
<td>Double</td>
<td>The satellite azimuth, in degrees</td>
</tr>
<tr>
<td>Double</td>
<td>The satellite elevation, in degrees</td>
</tr>
<tr>
<td>Int</td>
<td>The signal strength, in Db</td>
</tr>
<tr>
<td>Int</td>
<td>Whether this satellite is used (0 = No, Yes otherwise)</td>
</tr>
</tbody>
</table>

The operation will throw an exception if the index is invalid, it will return None if the current location module does not support satellite data.

### 3.12 LocationRequestor::Close()

Closes the connection previously opened using Open().

### 3.13 LocationRequestor::SetUpdateOptions(int updateInterval, int updateTimeout, int maxAge, int allowPartial)

Sets the update options for the NotifyPositionUpdate and/or InstallPositionCallback call. The update interval defines the interval between updates in seconds, updateTimeout defines the number of seconds after the call should time-out, maxAge specifies the maximum age (in seconds) of the returned location information, and allowPartial specifies whether partial location updates (containing a subset of the fields) are allowed (1) or not (0).

Note that 0 <= updateInterval < updateTimeout and 0 <= maxAge < updateInterval.

### 4 Example code

The code below tries to explicitly connect to the internal GPS, or if that is not available to an external (Bluetooth) GPS. Usually though that code can be skipped and lr.Open(-1) can be used: this will connect to the default location module.

```python
import time
import sys
import e32
import appuifw
import thread
import locationrequestor

def threadFunc():
    # create LocationRequestor instance
    lr = locationrequestor.LocationRequestor();
    # show default module
    print 'Default', lr.GetDefaultModuleId();
    # show number of modules
    count = lr.GetNumModules();
    print 'Count', count;
    # find internal GPS (if any)
    i = 0;
    id = -1;
    while i < count:
        info = lr.GetModuleInfoByIndex(i);
```
if (info[3] == locationrequestor.EDeviceInternal):
    id = info[0]
    print 'Using', info;

    i = i + 1;

# find external GPS (if any), if internal not found
if id < 0:
    i = 0;
    while i < count:
        info = lr.GetModuleInfoByIndex(i);
            id = info[0]
            print 'Using', info;

        i = i + 1;

# set update options
lr.SetUpdateOptions(3, 25, 2, 1);
# connect to position module
lr.Open(id);

# do a thousand updates
i = 0;
while i <= 1000:
    location = lr.NotifyPositionUpdate();
    # if full update then show, otherwise only show satellite data
    if (location[0] == 0):
        print 'Location', location
    else:
        print 'Sat', location[13], location[14]
    i = i + 1

# close connection to location module
print 'Calling close'
lr.Close()

print 'Done'

thread.start_new_thread(threadFunc,())