

mSense Middleware (Version 1.3.1)

mSense is a novel context acquisition and management middleware for high-end mobile phones based on Qt for S60. Within the middleware so called low-level sensor nodes encapsulate platform level APIs for seamlessly accessing hardware sensors, simulated sensor or web services, which are implicitly used as external data sources in the context of the middleware. High-level sensor nodes (so called channels or aggregators) combine information from other sensor nodes to acquire and extract particular sensor information and generate new knowledge. For example, GPS, an accelerometer and a compass could be combined to provide a more accurate inertial sensor node. For the moment the output is supplied via a native interface, but we will be working on an additional broadcast feature in the future to enable publication of the output data through a network interface or a web service. The current version of the mSense middleware incorporates following low- and high-level sensor nodes.

Low-level sensor nodes	High-level sensor nodes (channels)
Accelerometer: Sensor provides 3D acceleration of device at a configurable rate.	User Availability: Sensor provides information of the current user status (if the user is available or not). This is done by a combination of the calendar and device profile data.
Alarm Sensor: Sensor provides information as well as notifications of current active alarms.	Movement Status: Sensor provides information if the user is currently moving or not. This is done by a combination of the GPS location and the accelerometer sensor.
Calendar Sensor: Sensor provides current calendar data (meetings, etc.).	Position / Address: Sensor provides current position information of the user based on current GPS location. An additional a reverse geo-coder is used to retrieve the current position description (address data) for the GPS location.
Device Orientation: Sensor provides current device orientation and attitude (rotation information).	
GPS Location: Sensor provides GPS based location information of the device.	
Magnetic North: Sensor provides current heading of device with respect to magnetic north.	
Magnetometer: Sensor provides 3D geomagnetic flux density information.	
Profile Sensor: Sensor provides information about the current device profile.	
Weather Sensor: Sensor provides actual weather information via a web-service for a configurable area.	

Table 1: Available sensor nodes within the mSense middleware.

Additional information concerning implementation details is available from the enclosed source code documentation. A general overview of the system architecture and the basic concept behind mSense could be found in a separate paper of *Krösche et al.*¹.

¹ J. Krösche, A. Jakl, D. Gusenbauer, D. Rothbauer and B. Ehringer. *Managing context on a sensor enabled mobile device – the mSense approach*. IEEE International Conference on Wireless and Mobile Computing, Network and Communications (WiMob), pp. 135-140, 2009.

General Annotation on the mSense middleware:

The mSense middleware was initially started as a student's project with the ambition to develop a simple and generic interface and framework respectively for sensor data access and context acquisition as well as context management. For the moment we are working on several other projects based upon the mSense middleware. This also includes continuous extensions and improvements of mSense itself and therefore the current state of the middleware should be seen as snapshot of the whole development. In the near future we are intending to adapt following issues:

- Concrete implementation of an extended network interface for data publication. This network interface should enable the remote usage of the whole mSense middleware (like for example from other platforms like Java, Python etc.).
- Implementation of a more suitable facade instance for the whole middleware. This facade should act as single-point-of-entry interface for the middleware functionality instead of using the `<MSenseController>` instance.
- Implementation of sensor node configuration classes as generic data structures (similar to data classes).
- Implementation of new low- and high-level sensor nodes.
- Implementation of a common logging facility instead of using the default Qt debug output.
- Modularization and unification of error codes within the middleware.

Target Platforms / Platform Specifics:

- Qt 4.6.0
- S60 5th Edition Version 1.0
- Nokia N97 Version 1.0

We had to use Symbian S60 5th Edition platform specific code to implement low level sensor access through sensor APIs. This was done implicitly by using an adapted version of the Qt for S60 Mobile Extensions (3rd Technology Preview). This version of the Mobile Extension implementation offers additional sensor data and incorporates the magnetometer and magnetic north sensor.

The project was tested on the following devices:

- Nokia 5800 XpressMusic (without magnetometer and magnetic north sensor support)
- Nokia N97

NOTE: The mSense library was actually tested on the devices mentioned above as part of the mSense Demo-Application, which is available as enclosed project example (see *mSense Demo-Application*).

Built Instructions:

General Information: The mSense middleware is only available as dynamic library. This enables usage of the middleware in other projects and applications.

Project Dependencies: Due to the dependency to the Qt Mobility APIs the particular project include paths have to be adapted properly (by default it is assumed that the needed source and include files are located in a directory named *'qt-mobility-src-1.0.0-tp2'* at the same level as the project's root directory). The Qt Mobility APIs have to be installed for the particular SDK as well. The necessary source and include files of the Qt for S60 Mobile Extensions dependency are directly included in the project's source tree.

Symbian Capabilities: Due to the usage of the device's sensors, the location feature, network access and user data readouts the application needs at least *ReadDeviceData*, *WriteDeviceData*, *ReadUserData*, *WriteUserData*, *Location*, *NetworkServices* and *UserEnvironment* capability.

Google Reverse Geo-Coding: For reverse geo-coding (get position description or address data for the current GPS position) the Google Reverse Geo-Coding Service is used. Therefore a Google API Key is needed. You have to register your own Google API Key (<http://code.google.com/intl/en-EN/apis/maps/signup.html>) and define it in the global definitions file for the reverse geo-coding sensor *'src/sensors/reversegeocoder/ReverseGeocoderDefinitions.h'*).