KnowSensor X - Installation Guide and Architectural Overview

David Ganster

March 5, 2014

Contents

1 Disclaimer **2**

2 Requirements **2**

3 Installation **2**

3.1 Installation via the .mpkg file 2

3.2 Installation by compiling the code 2

3.2.1 Troubleshooting 3

4 Architectural Overview **4**

4.1 Quick start 4

4.2 Class diagram 5

4.3 Control/Data flow 6

4.3.1 Recording events and sending them to the server 6

4.3.2 Working with projects and activities 6

4.4 NSLogger 7

1 Disclaimer

This document is aimed at people familiar with the KnowSelf project [1], and does not go into detail about its core concepts - it merely describes the KnowSensor component for the Mac OS X platform.

2 Requirements

KnowSensor X will work on Macs running OS X Lion (10.7) or later, and has been thoroughly tested on Mavericks (10.9). Previous versions (10.6 “Snow Leopard” and below) are not supported yet, although the vast majority of the code base would not require any modifications to make previous versions compatible. The expected issues when compiling for 10.6 and below are nearly entirely confined to `.xib` files (interface builder layouts).

For KnowSensor X to work, you will need to have a binary installation of KnowSelf, which contains the KnowServer and KnowSelf web application (required to actually see the recorded events).

3 Installation

3.1 Installation via the .mpkg file

To install KnowSensor X on your Mac, simply doubleclick the installer file[1] and follow the on-screen instructions. The application (and both the KnowServer/KnowSelf web application) will be installed into the `Applications`[2] directory of the currently logged-in user (*not* in `/Applications/`!). The installation does not require a password and is specific to the current user, meaning that all users that want to use KnowSensor X must install the application individually (further implying that spying on other user’s activities is not possible due to the separate installation directories).

3.2 Installation by compiling the code

KnowSensor X does not rely on external third-party libraries, so compiling from source[3] should be as simple as opening the `KnowSensor X.xcodeproj` file in Xcode and building the target as usual.

[1]The installer is be available here: `http://know-center.tugraz.at/knowself/`

[2]The path to the installation directory is ’`<user name>/Applications/KnowSensorX/`’

[3]The sourcecode is available under [2]

If you are planning to use KnowSensor X with an external server, the resulting binary is all you need (don't forget to set the correct server address in the preferences!).

In the more likely case of a purely local installation though, you will still need to acquire a binary for the KnowServer/KnowSelf Web Application and arrange them in the following structure in the same directory that `KnowSensor X.app` resides in:

```
KnowSensor X.app
KnowSelf/
    KnowServer/
        bundles/
            ...
        mac.runner.args
        ...
    KnowSelf_WebApp/
        index.html
        AppController.j
        ...
```

After copying the files to the correct location, your application should work fine.

3.2.1 Troubleshooting

If you are having trouble compiling the code, here are a few tips:

- Upgrade your Xcode installation. `KnowSensor X` was tested on `LLVM 5.0` and Xcode 5.
- Confirm that you are using ARC, and disable it for the file `JSONKit.m` (you can do this by adding the `-fno-objc-arc` flag to this file in the build phases).
- Confirm that you are running at least Max OS X 10.7 (Lion) or above.
- If you are running an OS newer than 10.9 (Mavericks), check the code for deprecation warnings and set 10.9 as the base SDK.

4 Architectural Overview

This section will provide a general overview of the design of `KnowSensor X`, and is meant as a starting point for developers trying to get familiar with the codebase. A class diagram is provided to help understand relationships between classes at a glance in section 4.2. Technical information about the each individual class can be found in the `html/` folder accompanying this document.

Basic knowledge of Objective-C [3], MVC [4], Delegation (as commonly used in Apple's API [5]) and Grand Central Dispatch [6] is assumed. Detailed information about these topics can be found in Apple's technical documentation (links are at the end of this document).

All classes directly related to KnowSensor X are prefixed by '`KS`', to easily distinguish between internal classes and third-party objects.

4.1 Quick start

This is a short outline of the central classes that handle backend-logic. GUI classes are not mentioned here because their responsibilities are clearly defined by their name and they hardly contain any logic other than input handling.

- `KSSensor` - The base class for all sensors that are responsible for extracting data about the user's activity. It also defines a protocol that has to be implemented by objects that want to handle the recorded events.
- `KSSensorController` - A singleton class responsible for handling data recorded by the sensors. It creates and holds a reference to all sensor objects that are in use, and implements the event-handling protocol defined by `KSSensor`.
- `KSProjectController` - A singleton class that handles maintaining the active project/activity list as returned by the server. It is able to create projects/activities on the server, as well as changing and retrieving the currently recording activity.
- `KSUserInfo` - Another singleton, responsible for storing (and giving access to) all settings that can be changed by the user. Should you want to extend the GUI with new options, this is the place to store the associated data. Export/Import of settings to/from a file is also dealt with here.
- `KSAPIClient` - The high-level front end to the KnowServer's API. Its public interface is intentionally without any traces of network calls (such as JSON-responses), hiding all network-related code from the developer. It uses a block-based API to provide callbacks when network calls return.

4.2 Class diagram

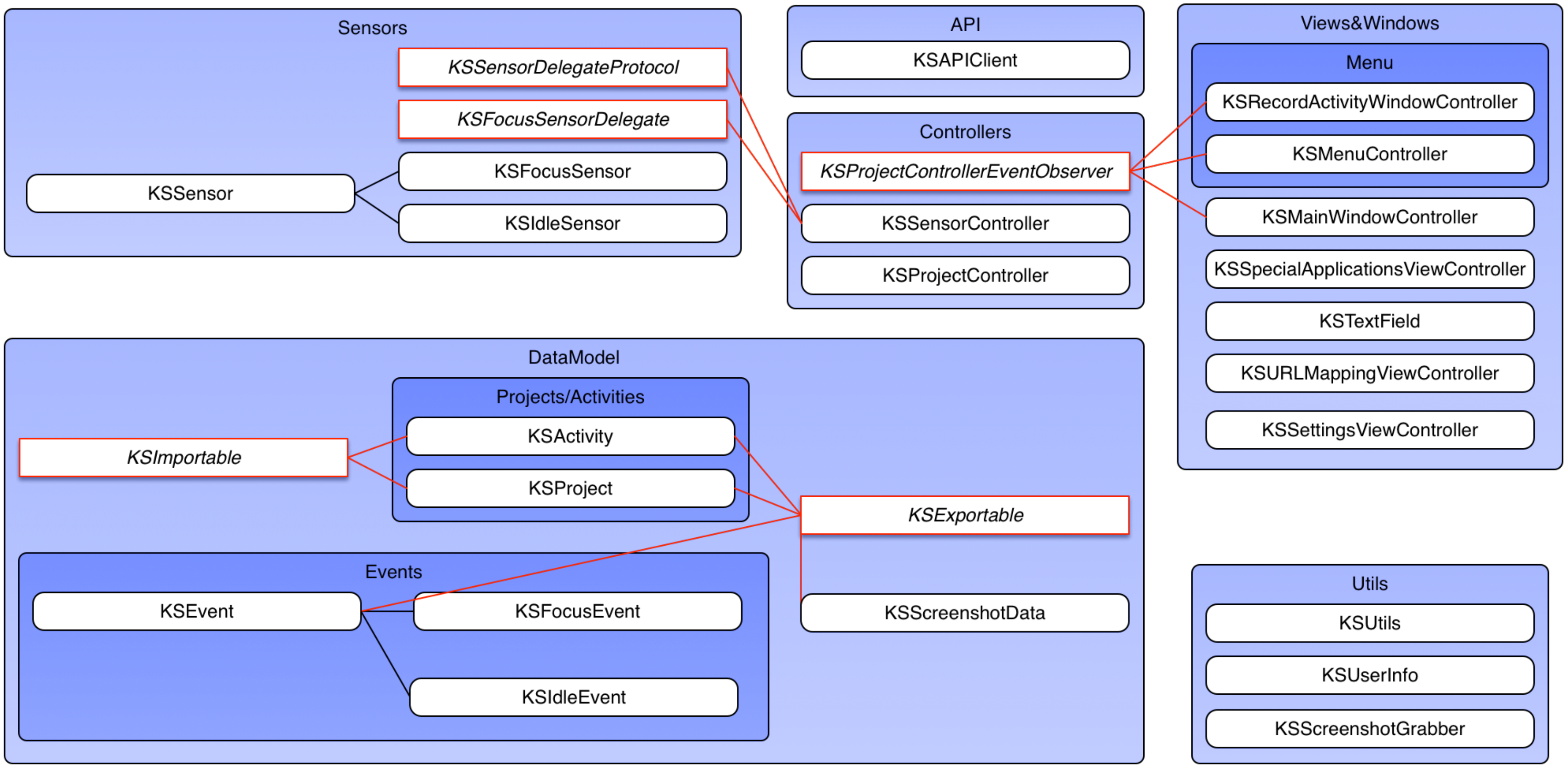

Figure 1: Red lines denote implemented protocols, black lines indicate inheritance. The grouping only represents logical connections (e.g. shared responsibilities), and does not reflect formal relationships like inheritance or composition.

4.3 Control/Data flow

Both the control- and data flow for all GUI-related classes are very straightforward and do not require an explanation additional to the documentation because of their self-contained nature. Handling the recording of events and activities involves a number of classes, on the other hand, which is why these processes are explained in more detail below.

4.3.1 Recording events and sending them to the server

Before events can be recorded, the `KSSensorController` has to be sent a '`startRecordingEvents`' message, which will in turn forward this message to all its sensors. This will cause all sensors to try and register their respective event listeners, and, after successful completion, start generating events.

Upon creation of an event, the `KSSensor` has to hand it to its delegate (which is the `KSSensorController`) for further processing. Once the `KSSensorController` receives an event, this object will be put at the end of a first-in-first-out buffer queue. Said queue contains all events in the order they were recorded - it is important that the server receives events in the order in the correct order (e.g. one must not send a 'user idle end'-event before starting idle). This requirement implies that all sensors must hand the recorded events to the delegate *in the correct order*. The process of emptying the queue conists of sending an event to the server (using the public interface of `KSAPIClient`), waiting for a positive response by the server, removing the just-sent object from the queue and sending the next event.

In case of an error, the `KSSensorController` will stop trying to empty to send events until the server is available again (by listening to a notification generated by `KSAPIClient`), or until the next event is put into the queue, in which case it will simply retry sending the first event immediately to prevent the buffer from growing too large.

All of this is done on a separate serial dispatch-queue, to enable the sensor that recorded the event to continue with its execution, and synchronize access to the event buffer queue. It is possible to specify a block that will be called once a specific event has successfully been sent to the server. This is especially useful when terminating the application, where the server can only be shut down after the last event has been sent (but the mechanism can of course be used for different kinds of synchronization as well).

4.3.2 Working with projects and activities

There are multiple GUI elements (and potentially other classes) that require access (and are interested in updates) to the project/activity list; interested objects can add themselves as an observer to the `KSProjectController`. The controller will then update all of its observers whenever the projects, activities or currently recording activity changed, telling each object exactly what has changed since the last update, while also providing read-access to the full project list.

The process of updating the lists themselves is completely opaque to any class outside the `KSProjectController`, the only required interaction with the controller is to send it a '`startUpdatingProjectList`' message once. Internally, the `KSProjectController` will poll the server[4] for all projects/activities and parse these lists only if *both calls returned successfully.* Otherwise, another poll will be scheduled when the server becomes available again.

Creating a new project/activity object can safely be done in any class (does not require a factory-method in `KSProjectController`), but the newly created object will not be uploaded until `createProject:`[5] (or `startRecordingActivity:` for activities, respectively) is called. **Warning:** If you want to start recording an activity for a new project, you will have to create the project on the server first. The `KSProjectController` will **not** do this for you!

Deleting projects or activities is not supported by the KnowServer's API at the time of writing, but the mechanisms for correctly handling deleted projects are mostly in place. Some synchronization mechanisms may need to be applied for UI classes that interact with the `KSProjectController` (i.e. `KSRecordActivityWindowController`).

4.4 NSLogger

Instead of using the standard `NSLog()` call to write messages to the console, `KnowSensor X` uses a more modern approach in NSLogger[7]. In order to see any log messages at all, you will need to download and compile NSLogger once, then launch it whenever you want to see debug-messages generated by `KnowSensor X`. The framework allows logging over a local network, using Bonjour discovery. The first computer in the subnetwork that is found to have NSLogger installed and open will receive log messages from any computer in the network (on which `KnowSensor X` is running, of course). This is extremely useful for debugging. Log messages will be buffered until a client is found in the network, which is why the release-version of `KnowSensor X` is very sparse when it comes to logging (basically all logs except for errors are disabled). Other benefits of NSLogger include the fine granularity with which log messages can be filtered - for example, every message has a domain and level associated with it, telling the reader precisely in which part of the program something was logged, as well as how severe the message is (e.g. error, exception, warning, info etc).

[4]The server interaction is handled through `KSAPIClient` of course.

[5]Another variant of this method called `createProject:success:failure:` can be used to be notified when the server acknowledged the creation.

References

[1] KnowSelf on the KnowCenter's homepage:
`http://know-center.tugraz.at/knowself/`

[2] The sourcecode for `KnowSensor X` is available under the MIT license here:
`http://github.com/davidganster/KnowSensorX`

[3] Objective-C quick-start guide for programmers:
`http://cocoadevcentral.com/d/learn_objectivec/`

[4] Model-View-Controller reference on Apple's documentation:
`https://developer.apple.com/library/mac/documentation/general/conceptual/devpedia-cocoacore/MVC.html`

[5] Delegation Pattern on Apple's documentation:
`https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html`

[6] Grand Central Dispatch documentation:
`https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/Reference/reference.html`

[7] NSLogger on Github:
`http://github.com/fpillet/NSLogger`