Java ME Mail API
Design Document

Last modified: September 11, 2009

Changes History

Date	*Author*	*Changes*
2008.12.12	Marco Garatti	Initial Draft
07/21/09	Giulia Zanchi	Fixed copyright info and front page

Table of Contents

1. Overview

The Funambol PIM API allows application developers to synchronize PIM data using standard methods to access PIM data. This API contains classes and methods that implement sync sources to synchronize PIM data using JSR75. The API relies on Funambol SyncML API to handle the SyncML synchronization and on JSR75 to access PIM data on devices. The API also depends on the Funambol common API for basic functionalities.

This document explains, from a developer point of view, the architecture of the Funambol PIM API for Java 2 Micro Edition.

1.1. Document Conventions

The diagrams used in this document are inspired to the UML sequence and class diagrams, but with some simplification. The conventions used by the diagrams are described in the following sections.

1.1.1. Sequence Diagrams

- Each entity is represented as a box;
- a box can represent a class, an instance, an interface or even a conceptual entity; the real meaning depends by the context;
- solid arrows represents method or function calls;
- dashed arrows represent some sort of communication between two entities; it is intended that the communication mechanism is left unspecified or is not important or it is at a different abstraction layer.

1.1.2. Class Diagrams

- Each class is represented as a box;
- data members and methods are separated by a horizontal line;
- plain titles represent classes, italicized titles represent interfaces (abstract classes);
- a + next to a method or data member name means “public”
- a - next to a method or data member name means “private”
- a # next to a method or data member name means “protected”;
- a > next to a data member name means it is a property with get/set accessors;
- inheritance is represented by an arrow pointing to the base class (empty arrow)
- class usage is represented by an arrow pointing to the used class (filled arrow)
- italicized methods names represent abstract methods.

1.2. Overview

The Funambol PIM API for J2ME is basically a set of sync sources that allow the synchronization of PIM data. These sync sources are designed to be largely modular and customizable, but at the same time the are a ready to use component that clients can use directly.

A client can use the available sync sources and start to synchronize PIM data right away. Or it can customize the behavior in many different ways. For example:

- using a custom changes tracking mechanis
- using custom formats to exchange data
- extending basic formats to support custom fields

This API relies on other APIs and the reader should be familiar with these other APIs to fully understand this document.

Funambol Common API is described in [10], Funambol SyncML API in [] and JSR75 in [].

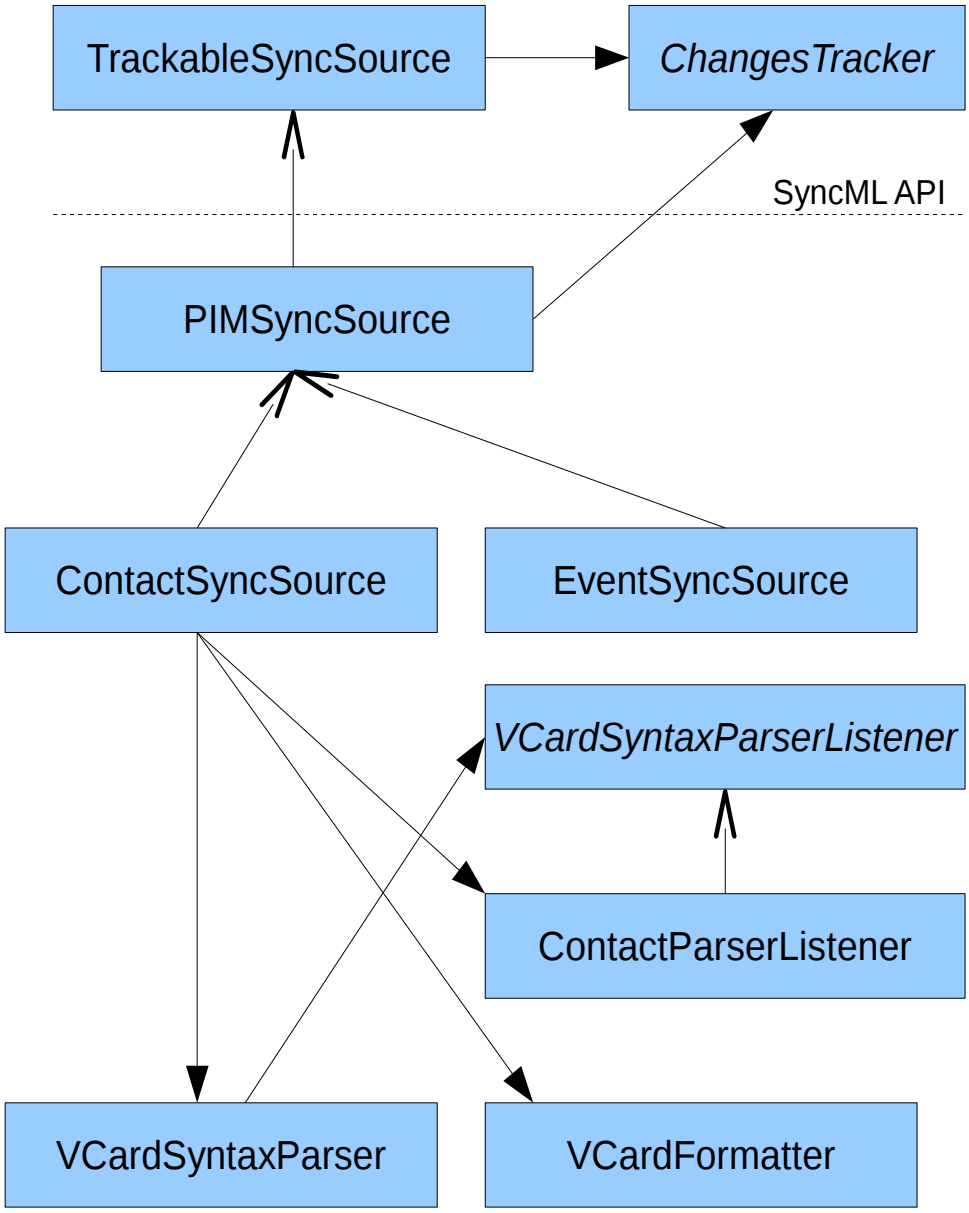
2. PIM module architecture

As anticipated in the overview this module contains the implementation of customizable sync source. Figure ?? shows the class diagram where a ContactSyncSource is fully described, while others are only partially described (EventSyncSource) or not described at all (TaskSyncSource and NoteSyncSource). The classes that are not described or only partially described are implemented using the same pattern used for contacts. For the sake of simplicity they were left out.

The module provides its main functionalities through the following classes:

- ContactSyncSource, EventSyncSource, TaskSyncSource, NoteSyncSource are the PIM sync sources
- VcardSyntaxParser, ContactParserListener and VcardFormatter are vCard parser and formatter
- Parsers and formatters for all other types (TBD)

The following chapter describes into more details these classes and their role.


TrackableSyncSource
ChangesTracker
SyncML API
PIMSyncSource
ContactSyncSource
EventSyncSource
VCardSyntaxParserListener
ContactParserListener
VCardSyntaxParser
VCardFormatter

3. Classes description

This chapter provides a high level description of the main classes provided by the PIM module.

3.1. Sync sources implementation

Four classes implement sync sources for PIM synchronization. All these classes inherit from a PIMSyncSource which has generic methods shared among the subclasses. The PIMSyncSource is an abstract implementation of TrackableSyncSource (see ?syncml ref?). As such it can be configured with an arbitrary ChangesTracker to keep track of the changes since the last synchronization. This sync source provides the following main methods:

- addItem, updateItem and deleteItem invoked on item's commands from the server
- getAllItemsKeys used by finger print based tracker to compute the set of changes or for slow synchronizations. Tracker non fingerprints based (e.g. based on listener) may avoid invoking this method for fast sync, increasing the source efficiency
- getItemContent used to fetch an item content

All these methods work in terms of PIMItem and are therefore generic as they can be used with any JSR75 type of data.

The source is abstract because it needs functionalities specific to particular data type. The following methods are what concrete implementations must provide:

- create and delete a single item. The operation must be performed on a concrete PIMList and needs to be implemented in each sync source
- fill and format an item. These methods convert a PIM datum from and to a serializable representation. For example contacts by default are exchanged as vCard
- get supported fields and ID field. These are helper methods used to perform various operations

Each concrete implementation has to provide these methods.

It is important to note that the default implementation in the module use standard format to exchange data. In particular:

- Contacts are exchanged as vCard (2.1)
- Events
- Tasks (TBD)
- Notes (TBD)

If a client needs to change this behavior, it can simply inherit from the appropriate SyncSource and redefine the methods that create and format items.

The tracking mechanism is a parameter for each source. There is no default behavior, but a client may use the CacheTracker to get a working implementation in no time. Trackers are described in [?funambol syncml?].

3.2. Parsers and formatters

This is the second main functionality provided by this module. The ability to convert an item in text and viceversa where “text” stands for a standard format such as vCard.

Formatters are simple classes that take a PIMItem (of a given concrete type) and format it on an OutputStream. Each formatter can format items dumping all fields (including empty ones) or only the ones with some data. This is useful to handle updated items versus new items.

Parsers are a little more complicated. First of all because parsing is more complex than formatting, but also because to make the structure flexible different classes were introduced.
First of all a concept of syntax parser was introduced. Such a parser scans an input data stream and performs a syntactical analysis. Semantics actions are completely decoupled and implemented in listeners. When the parser recognizes a part of information (e.g. the name of a contact) it invokes the listener with all the necessary information. The listener can use this information to fill a PIMItem.
One example is the parsing mechanism for vCards. The VcardSyntaxParser is a generic parser shared between the pim-framework and client api. The parser is specified in JavaCC and it parses vCard 2.1. It uses a VcardSyntaxParserListener generic interface to notify the recognition of vcard fields. The PIM module has an implementation of this interface for vCard (ContactParserListener) that builds a JSR75 Contact. This implementation is very flexible as it could allow the following things:

- A SIF-C parser could use the same listener to fill a JSR75 contact
- A different implementation of the listener allows the creation of a different type of data. On platforms where JSR75 is not available, or it is preferable to use a different data model, it is possible to rewrite the listener while using the parser.

4. References

[OMA-DS] SyncML Data Synchronization Protocol, version 1.2, Open Mobile Alliance
[COMMON] Funambol J2ME Common API Design Document, version 1.0, Funambol Inc.

[API-J2SE] Funambol Java API J2SE Developer Guide, version 1.0, Funambol Inc.
[API-CPP] Funambol 3.0 Client API C++ Design Document, version 1.0, Funambol Inc.