Funambol DS Server

SyncSource API

Version 3.0
May 2006

Important Information

Published by Funambol, Inc., 643 Bair Island Road, Suite 305, Redwood City, CA 94063

Contents

Overview

This document describes the interface for developers who wish to develop a SyncSource. We will use the following terms and concepts:

Module: a container for anything related to a Funambol DS Server extension. For example, a module may consist of a packaged set of files, including classes, configuration files, server beans, and initialization SQL scripts, that are embedded into the Funambol DS Server to provide access to a specific database for data synchronization.

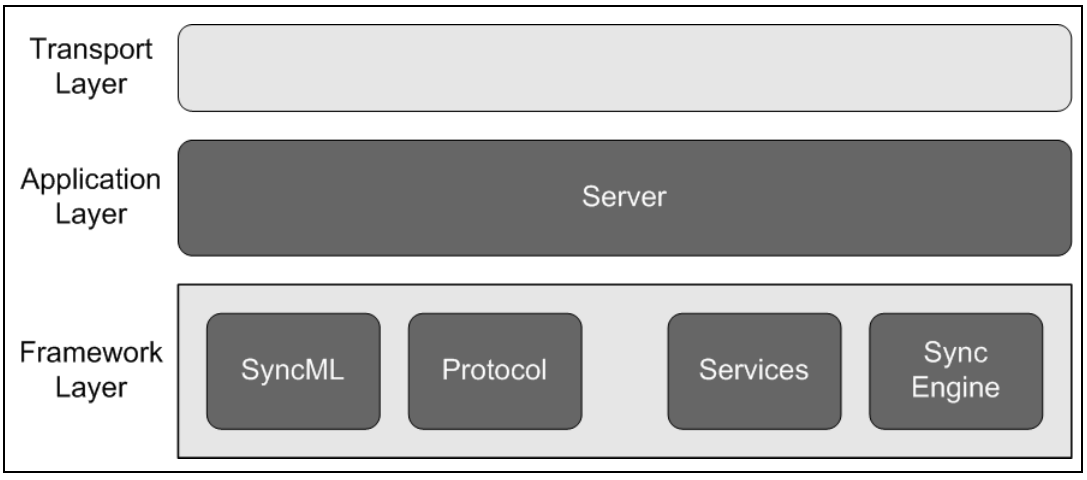
Connector: a server extension that provides support for data synchronization with a specific data source. For example, the Funambol DB Connector provides a GUI and runtime classes for the synchronization of generic data stored in a RDMS. Alternatively, a Connector could support a data source that stores email messages, calendar events, or other data types.

SyncSource: a key component of a Connector, it defines the way a set of data is made accessible to the Funambol DS Server for synchronization. Funambol provides SyncSources for the common uses (such as for files), but also allows independently-developed SyncSources to be plugged into the synchronization engine, allowing synchronization with virtually any database or type of data.

This document provides reference information on the SyncSource interface and related classes. For background information on the synchronization process, see *Sync4j Architecture*. For instructions on packaging, installing and testing modules, see the *Funambol DS Server Module Development Tutorial*.

Server Architecture

The Funambol DS Server architecture is implemented in layers, as shown below:



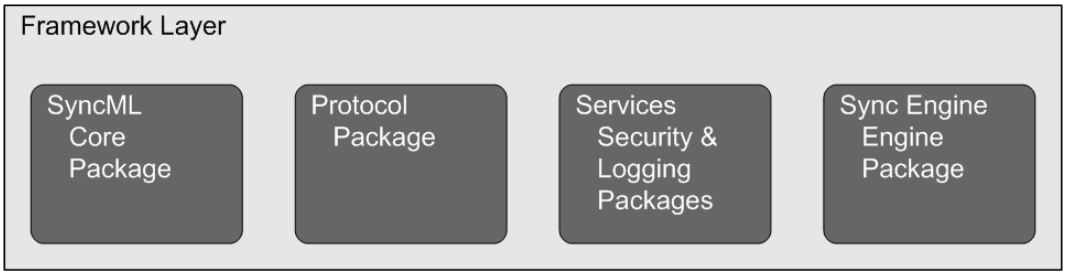
Transport Layer – the server relies on the transport layer to receive messages delivered using different protocols, such as HTTP, SMTP, OBEX, and so on. Currently, Funambol supports the HTTP protocol for message transport.

Application Layer – consists of the server that accepts and processes SyncML messages. The server is implemented as a J2EE application that can be deployed in a J2EE-compliant application server.

Framework Layer – implements and provides protocols, horizontal services, and the synchronization engine.

Framework Layer Packages

The framework layer consists of several packages, the most important of which are shown below:



Core Package – the foundation classes used to represent a message. This module provides for the translation of an XML stream into an object tree, and an object representing a message can be converted into the corresponding XML representation.

Protocol Package – a SyncML communication is a sequence of correlated messages that must follow additional rules, as specified by the SyncML protocol. This module handles the processing of such messages to ensure compliance with the protocol.

Security and Logging Packages – implements logging and security services.

Engine Package – provides the logic for the synchronization server, including the following:

- Identify the source and destination of the data to be synchronized.
- Identify the data that needs to be updated, added, or deleted.
- Determine how updates are to be applied.
- Detect and resolve conflicts.

The Engine Package

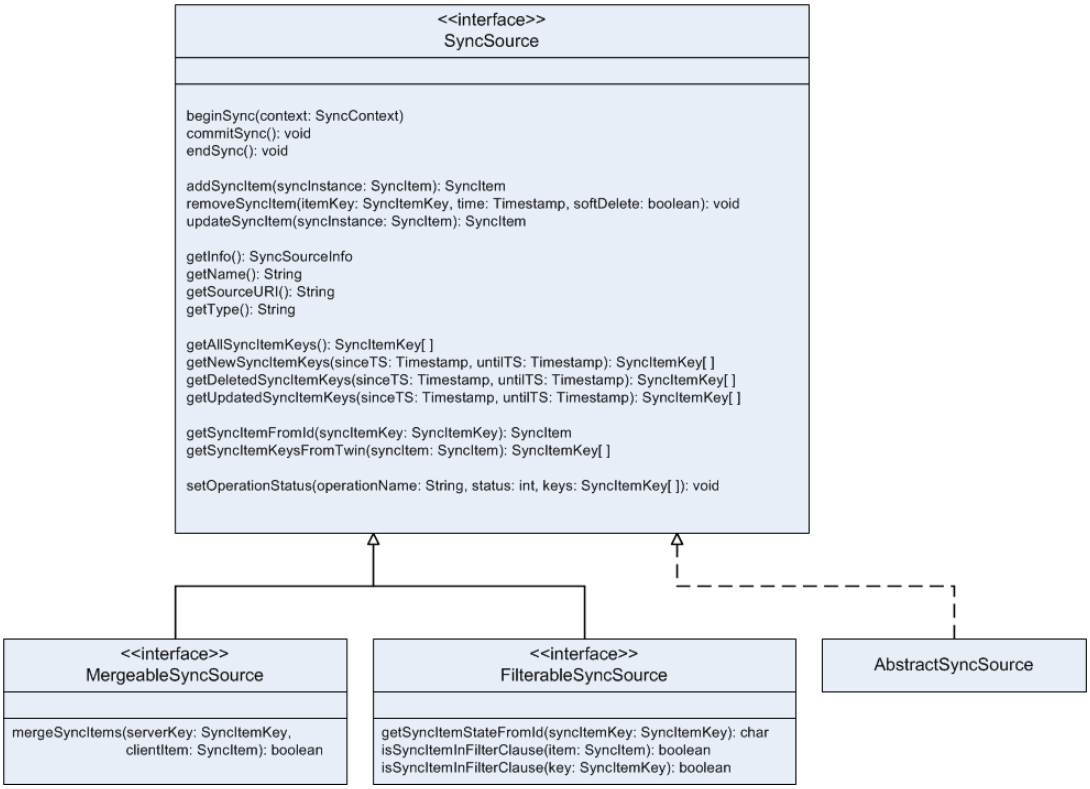
The engine package provides an interface for the synchronization engine. The basic interface and classes are grouped in the package *com.funambol.framework.engine*. The reference implementation is grouped under *com.funambol.server.engine*. The core of the SyncSource architecture is the interface *com.funambol.framework.engine.source.SyncSource*.

The SyncSource Interface

A SyncSource defines the way a set of data is made accessible to the Funambol DS Server for synchronization. The SyncSource interface does not make any assumptions about the type of data to be synchronized.

A SyncSource is identified by a unique domain-specific name and a unique sourceURI, which is the URI that a SyncML client must specify to synchronize with the SyncSource. A SyncSource is also associated with a MIME type that specifies the type of data handled.

The core of the SyncSource is the *com.funambol.framework.engine.source.SyncSource* interface.



Funambol provides two extensions of the SyncSource interface named *MergeableSyncSource* and *FilterableSyncSource*, as well as an abstract implementation *AbstractSyncSource*.

MergeableSyncSource

You use the *com.funambol.framework.engine.source.MergeableSyncSource* interface when a data source wants to support a conflict resolution strategy based on merging the conflicting data. When a conflict is detected, a merge is performed to avoid loss of information. For example, suppose the email address for a contact is updated on the server, and the phone number for the same contact is updated on a client. In this case, the server contact data and the client contact data can be merged. For additional details, see “MergeableSyncSource” on page 12.

FilterableSyncSource

You use the *com.funambol.framework.engine.source.FilterableSyncSource* interface when a SyncSource supports a filter as specified in the SyncML 1.2 protocol. The protocol specifies the following types of filters that can be used by a device:

Record filter – specify the records to be synchronized, e.g., today's email only.

Field filter – specify roles on the fields to be synchronized, e.g., no contact photos.

In addition, a filter can be specified as EXCLUSIVE or INCLUSIVE. When using an EXCLUSIVE filter, the server deletes all items on the client that are not included in the filter. When using an INCLUSIVE filter, the client deletes its items that are not included in the filter, and sends the delete command to the server (soft delete).

For additional details, see "FilterableSyncSource" on page 12.

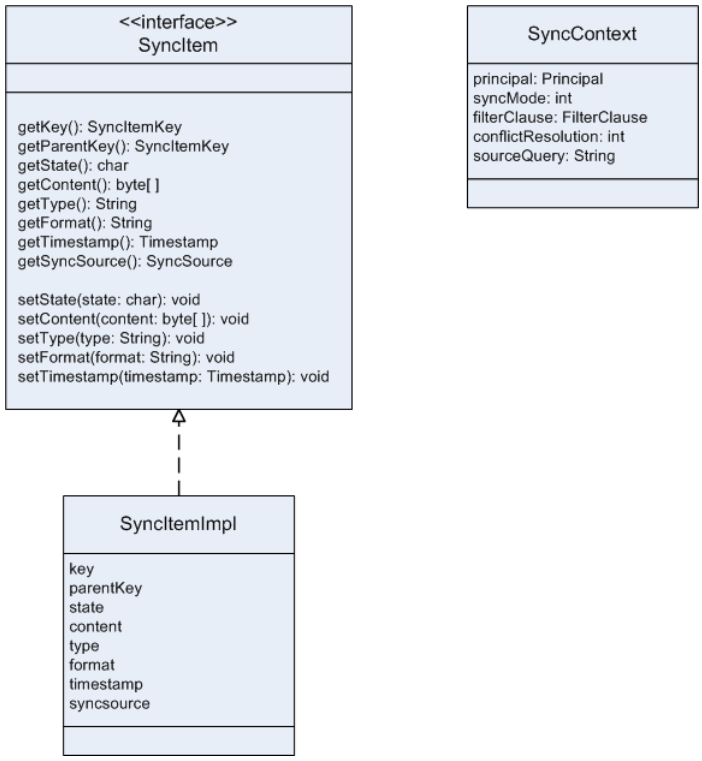
AbstractSyncSource

This is a utility class that implements the most common methods of the SyncSource interface, but delegates the methods required to access the data source to concrete extensions. The following are the defined abstract methods:

- getAllSyncItemKeys
- getDeletedSyncItemKeys
- getNewSyncItemKeys
- getSyncItemFromId
- removeSyncItem
- updateSyncItem
- addSyncItem
- getSyncItemKeysFromTwin
- setOperationStatus

Related Classes

Classes related to the SyncSource interface include the SyncItem interface and the SyncContext and SyncItemImp classes.



SyncItem Interface

The items returned by a SyncSource are *com.funambol.framework.engine.SyncItem* objects. A SyncItem is the indivisible entity that can be exchanged in a synchronization. It is identified by a unique SyncItemKey, and also is associated with a state. A SyncItem has the following properties:

- state
- content
- type
- format
- timestamp

Funambol provides a standard implementation of the SyncItem interface in the class *com.funambol.framework.engine.SyncItemImpl*. For details on the methods defined in the SyncItem interface, see “SyncItem” on page 13.

SyncContext

A new synchronization is started by calling the `beginSync` method with SyncContext as the input parameter (for details, see "beginSync" on page 7). A SyncContext contains the following information:

principal – the entity that is requesting the synchronization; it represents the combination of a user and a device and can be used by the SyncSource to consider only the data related to this entity, such as the contacts of a particular user. If null, all items in the data source are considered for synchronization, regardless of the principal to which they belong.

syncMode – the type of synchronization, specified as follows:

syncMode	Type of Synchronization
200	TWO_WAY
201	SLOW
202	ONE_WAY_FROM_CLIENT
203	REFRESH_FROM_CLIENT
204	ONE_WAY_FROM_SERVER
205	REFRESH_FROM_SERVER

filter – the filter specified by the client. If specified, and the SyncSource is FilterableSyncSource, the filter provides the following roles:

- The `getAllSyncItemKeys`, `getDeletedSyncItemKeys`, `getNewSyncItemKeys`, and `getUpdatedSyncItemKeys` methods return only the items that satisfy the filter criteria.
- The `getSyncItemsFromId` and `getSyncItemsFromTwin` methods ignore the filter.

For additional details, see "FilterableSyncSource" on page 4.

conflictResolution – specifies the conflict resolution used by the server in the synchronization. Valid values:

- `SyncContext.CONFLICT_RESOLUTION_SERVER_WINS` – conflicts resolved with server data.
- `SyncContext.CONFLICT_RESOLUTION_CLIENT_WINS` – conflicts resolved with client data.
- `SyncContext.CONFLICT_RESOLUTION_MERGE_DATA` – conflicts are resolved by merging server and client data (see "MergeableSyncSource" on page 12).

sourceQuery – the query string specified by the client in the remote database name. For example, if the following is specified: `'cal?param1=value1&param2=value2'` the source query is `param1=value1&param2=value2`.

SyncSource

The SyncSource interface defines the following methods:

addSyncItem

Declaration `public SyncItem addSyncItem(SyncItem syncInstance)`

Description Called by the engine to add a new SyncItem. The item is also returned, which enables the source to modify its content and return the updated item (e.g., updating the id to the GUID).

Parameters `syncInstance` – The item to add.

Throws `SyncSourceException`

beginSync

Declaration `public void beginSync(SyncContext syncContext)`

Description Called by the engine to start a new synchronization with the specified SyncContext.

Parameters `syncContext` – contains the principal, syncMode, and required filter. For details, see "SyncContext" on page 6.

Throws `SyncSourceException`

commitSync

Declaration `public void commitSync()`

Description Called by the engine to commit the changes applied during the synchronization session. The method is called for each SyncML message processed.

Throws `SyncSourceException`

endSync

Declaration `public void endSync()`

Description Called by the engine after the modifications have been applied.

Throws `SyncSourceException`

getAllSyncItemKeys

Declaration

```
public SyncItemKey[] getAllSyncItemKeys()
```

Description Called by the engine to get the SyncItemKeys of all items based on the parameters used in the `beginSync` call. Returns an array of the SyncItemKeys stored in this source. Returns an empty array if there are no items.

Throws `SyncSourceException`

getDeletedSyncItemKeys

Declaration

```
public SyncItemKey[] getDeletedSyncItemKeys(Timestamp sinceTs,
                                            Timestamp untilTs)
```

Description Called by the engine to get the SyncItemKey of all items deleted during the time period `sinceTs` - `untilTs`. This time period is the time between the last synchronization and the start time of the current synchronization. If `sinceTs` is null, gets the SyncItemKey of all items deleted up to and including `untilTs`. If `untilTs` is null, gets the SyncItemKey of all items deleted from `sinceTs` and later.

Parameters `sinceTS` – beginning point of time period, i.e., for a fast synchronization, the time of the last synchronization. For a slow synchronization, this parameter is null.

`untilTS` – ending point of time period.

Throws `SyncSourceException`

getInfo

Declaration

```
public SyncSourceInfo getInfo()
```

Description Called by the engine to get the SyncSourceInfo of the source.

getName

Declaration

```
public String getName()
```

Description Called by the engine to get the name of the source.

getNewSyncItemKeys

Declaration
```
public SyncItemKey[] getNewSyncItemKeys(Timestamp sinceTs,
                                        Timestamp untilTs)
```

Description Called by the engine to get the SyncItemKey of the items created during the time period `sinceTs` - `untilTs`. This time period is the time between the last synchronization and the start time of the current synchronization. If `sinceTs` is null, gets the SyncItemKey of all items created up to and including `untilTs`. If `untilTs` is null, gets the SyncItemKey of all items created from `sinceTs` and later.

Parameters `sinceTS` – beginning point of time period, i.e., for a fast synchronization, the time of the last synchronization. For a slow synchronization, this parameter is null.

`untilTS` – ending point of time period.

Throws `SyncSourceException`

getSourceURI

Declaration
```
public String getSourceURI()
```

Description Called by the engine to get the source URI of the source.

getSyncItemFromId

Declaration
```
public SyncItem getSyncItemFromId(SyncItemKey syncItemKey)
```

Description Called by the engine to get the item with the specified key. If no item is found, returns null.

Parameters `syncItemKey` – the SyncItemKey of the item.

Throws `SyncSourceException`

getSyncItemKeysFromTwin

Declaration
```
public SyncItemKey[] getSyncItemKeysFromTwin(SyncItem syncItem)
```

Description Called by the engine to get the SyncItemKeys of the twins of the given item. Each source implementation can interpret this as desired (i.e., comparing all fields).

Parameters `syncItem` – the twin item.

Throws `SyncSourceException`

getType

Declaration `public String getType()`

Description Called by the engine to get the type of the source, e.g., text/x-vcard.

getUpdatedSyncItemKeys

Declaration
```
public SyncItemKey[] getUpdatedSyncItemKeys(Timestamp sinceTs,
                                            Timestamp untilTs)
```

Description Called by the engine to get the SyncItemKey of the items updated during the time period `sinceTs` - `untilTs`. This time period is the time between the last synchronization and the start time of the current synchronization. If `sinceTs` is null, gets the SyncItemKey of all items updated up to and including `untilTs`. If `untilTs` is null, gets the SyncItemKey of all items updated from `sinceTs` and later.

Parameters `sinceTS` – beginning point of time period, i.e., for a fast synchronization, the time of the last synchronization. For a slow synchronization, this parameter is null.

`untilTS` – ending point of time period.

Throws `SyncSourceException`

removeSyncItem

Declaration
```
public void removeSyncItem(SyncItemKey itemKey, Timestamp time,
                           boolean softDelete)
```

Description Called by the engine to remove (soft delete) a SyncItem that is identified by the specified key.

Parameters `itemKey` – the key of the item to remove.

`time` – the time of the deletion.

`softDelete` – specifies whether the removal is a soft delete.

Throws `SyncSourceException`

setOperationStatus

Declaration

```
public void setOperationStatus(String operationName, int status,
                               SyncItemKey[] keys)
```

Description Called by the engine to notify the status of an operation (Add/Replace/Delete) performed on the client.

Parameters `operationName` – the name of the operation. Valid values: Add, Replace, Delete.

`status` – the status of the operation.

`keys` – the SyncItemKeys of the items.

updateSyncItem

Declaration

```
public SyncItem updateSyncItem(SyncItem syncInstance)
```

Description Called by the engine to update a SyncItem. The item is also returned, which gives the source the opportunity to modify its content and return the updated item (e.g. update the id to the GUID).

Parameters `syncInstance` – The item to update.

Throws `SyncSourceException`

MergeableSyncSource

The MergeableSyncSource interface defines the following method:

mergeSyncItems

Declaration `public boolean mergeSyncItems(SyncItemKey serverKey, SyncItem clientItem)`

Description Called when a conflict must be resolved by merging items. On the server side, the result of the merge must be persistent in the underlying data source. If the item on the client is to be updated, this method returns "true" and puts the new content in the specified `clientItem`.

Parameters `serverKey` – the item's key on the server.

`clientItem` – the item on the client.

Throws `SyncSourceException`

FilterableSyncSource

The FilterableSyncSource interface defines the following methods:

getSyncItemStateFromId

Declaration `public char getSyncItemStateFromId(SyncItemKey syncItemKey)`

Description Called by the engine to get the status of the item with the specified key. Used to discover the status of an item that is not in the filter criteria. If no item is found, returns `SyncItemState.NOT_EXISTING`.

Parameters `syncItemKey` – the SyncItemKey of the item.

Throws `SyncSourceException`

isSyncItemInFilterClause

Declaration `public boolean isSyncItemInFilterClause(SyncItem item)`

Description Called by the engine to check if the specified item satisfies the filter clause specified in the `beginSync` call. Returns true if the item satisfies the filter, false otherwise.

Parameters `item` – the item to check.

Throws `SyncSourceException`

isSyncItemInFilterClause

Declaration `public boolean isSyncItemInFilterClause(SyncItemKey key)`

Description Called by the engine to check if the item with the specified key satisfies the filter clause specified in the `beginSync` call. Returns true if the item satisfies the filter, false otherwise.

Parameters `key` – the key of the item to check.

Throws `SyncSourceException`

SyncItem

The SyncItem interface defines the following methods:

getContent

Declaration `public byte[] getContent()`

Description Called to get the SyncItem's content.

getFormat

Declaration `public String getFormat()`

Description Called to get the SyncItem's format.

getKey

Declaration `public SyncItemKey getKey()`

Description Called to get the SyncItem's key (i.e., unique identifier).

getParentKey

Declaration `public SyncItemKey getParentKey()`

Description Called to get the SyncItem's parent key.

getState

Declaration `public char getState()`

Description Called to get the SyncItem's state.

getSyncSource

Declaration `public SyncSource getSyncSource()`

Description Called to get the SyncItem's SyncSource.

getTimestamp

Declaration `public Timestamp getTimestamp()`

Description Called to get the SyncItem's timestamp.

getType

Declaration `public String getType()`

Description Called to get the type of the SyncItem's content.

setContent

Declaration `public void setContent(byte[] content)`

Description Called to set the SyncItem's content.

Parameters `content` – the content to set.

setFormat

Declaration `public void setFormat(String format)`

Description Called to set the SyncItem's format.

Parameters `format` – the format to set.

setState

Declaration `public void setState(char state)`

Description Called to set the SyncItem's state.

Parameters `state` – the state to set.

setTimestamp

Declaration `public void setTimestamp(Timestamp timestamp)`

Description Called to set the SyncItem's timestamp.

Parameters `timestamp` – the timestamp to set.

setType

Declaration `public void setType(String type)`

Description Called to set the type of the SyncItem's content.

Parameters `type` – the type to set.

Resources

This section lists resources you may find useful.

Related Documentation

This section lists documentation resources you may find useful.

Funambol DS Server Documentation

The following documents form the Funambol DS Server documentation set:

- *Funambol DS Server Architectural Overview*: Read this document for an overview of the architecture.
- *Funambol DS Server Administration Guide*: Read this guide to gain an understanding of installation, configuration, and administration.
- *Funambol DS Server Developer's Guide*: Read this guide to understand how to develop extensions to the server.
- *Funambol DS Server SyncSource API*: This document.
- *Funambol DS Server Quick Start Guide*: Read this guide to install and run a simple demonstration of synchronizing PIM data using the Funambol DS Server.
- *Funambol DS Server Module Development Tutorial*: Read this tutorial for instructions on packaging, installing and testing modules.