# Funambol JSON Connector Developer's Guide

DRAFT

*Last modified: July 28, 2010*

# Table of Contents

# 1. Introduction

This document describes how to install and manage the JSON Connector component for the Funambol Data Synchronization Service using the Administration Tool console. It also provides a detailed description of the REST API that the connector uses to communicate with an external PIM data source.

This document is intended for administrators and developers.

# 2. Installing the Funambol JSON Connector

## 2.1. Installation steps

Installing and configuring the Funambol JSON Connector requires a number of steps, described in summary in the following list:

1. Check if there is an available database connection (MySQL, PostgreSQL, Hypersonic).
2. Start the Data Synchronization Service (for more information, see [2]).
3. Install the Funambol JSON Connector (see section 2.2).
   - The installation procedure automatically creates the Connector database schema.
4. Run the Funambol Administration Tool (for more information, see [2]).
5. Configure the JSON Connector's properties (see section 2.3).
   - Set the back-end URL in the JSON Connector Panel of the Funambol Administration Tool.
6. Check the JSON SyncSources.
   - Create and set the parameters in the JSON SyncSource Panel of the Funambol Administration Tool.
7. Configure the JSON Officer in the Server Settings Panel of the Funambol Administration Tool.
8. Set the Encryption Level in the Funambol Administration Tool *(optional)*.
9. Set the Log Level in the Funambol Administration Tool *(optional)*.
10. Start a sync session from your mobile device.

## 2.2. Installing the Funambol JSON Connector

The Funambol JSON Connector is distributed as a standard Funambol module. The distribution contains the following files:

- *funambol-json-<major>.<minor>.<build number>.s4j* (the module)
- Release notes
- *readme.txt*
- Funambol JSON Connector Developer's Guide

To install the connector, follow these steps:

1. Copy the *.s4j* file in the directory *$FUNAMBOL_HOME/ds-server/modules.*
2. Modify the *install.properties* file adding *funambol-json-*.*.** to the modules list:

```
modules-to-install=foundation-7.x.x,...,funambol-json-*.*.*
```

3. Set the *JAVA_HOME* and the *J2EE_HOME* paths.
4. Go to the directory *$FUNAMBOL_HOME/Funambol/bin* and call the module installation command:

```
./install-modules
```

During the installation procedure, the following steps are performed automatically:

- The database is initialized, connector-specific tables are created and the connector is registered into the server.
- The *JSONOfficer.xml* file is copied in the directory

  *$FUNAMBOL_HOME/ds-server/config/com/funambol/server/security*

If you are upgrading from a previous version, please read the chapter 4.

## 2.3. Configuring the Funambol JSON Connector

Once installation is complete, you can use the Funambol Administration Tool to configure the JSON Connector.

In the left pane, expand the *Modules* tree and then select *jsonconnector* (see Figure 1).

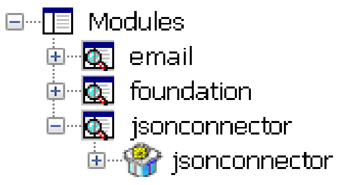
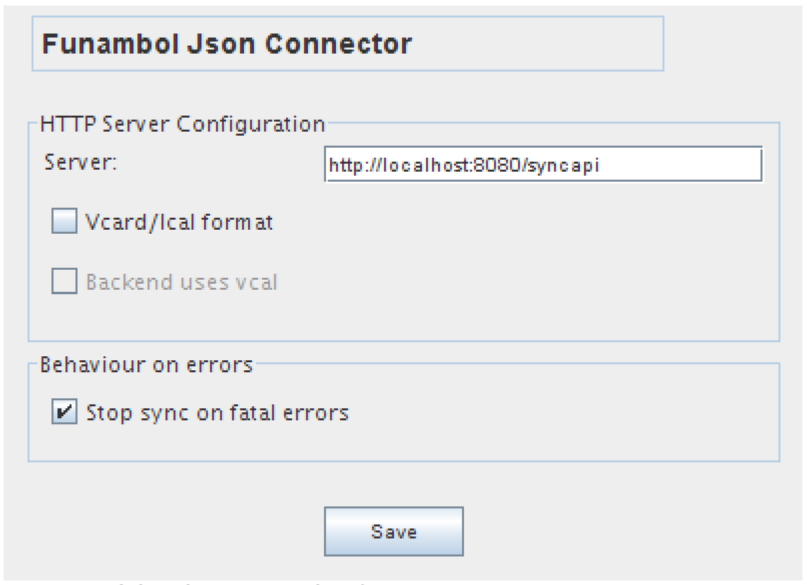


*Figure 1: Modules tree*

The JSON Connector Configuration Panel will appear in the right pane (see Figure 2).


Funambol Json Connector

HTTP Server Configuration

Server: http://localhost:8080/syncapi

☐ Vcard/Ical format

☐ Backend uses vcal

Behaviour on errors

☑ Stop sync on fatal errors

*Figure 2: JSON Connector Configuration Panel*

The JSON Connector properties that can be set are listed in the following table:

| ***Property*** | ***Description*** |
|---|---|
| URL | The address of the back-end server |
| Stop sync on fatal errors | When this flag is checked, if the backend returns a 500 error code for an item, the sync is stopped, otherwise the normal behavior of rejecting the command to the client is maintained. |

## 2.4. Configuring the SyncSources

In order to set up the JSON Connector SyncSources, go to the Administration Tool Console and expand the tree structure.

The Contact SyncSources have the following properties:

| ***Property*** | ***Description*** |
|---|---|
| Source URI | The SyncSource URI [e.g. “card” ]. |
| Name | The SyncSource name. [e.g.  “card” ] |
| Client Type | The preferred content type used talking with the Syncml clients |
| Datastore Tye | The content type used talking with talking with the datastore (json-extended/vcard) |

The Calendar/Task SyncSources have the following properties:

| ***Property*** | ***Description*** |
|---|---|
| Source URI | The SyncSource URI [e.g. “cal” ]. |
| Name | The SyncSource name. [e.g.  “cal” ] |
| Client Type | The preferred content type used talking with the Syncml clients |
| Datastore Type | The content type used talking with talking with the datastore (json-extended / vcal / ical) |
| Subtype | Whether to sync only events, tasks or both |

The Note SyncSources have the following properties:

| ***Property*** | ***Description*** |
|---|---|
| Source URI | The SyncSource URI [e.g. “note” ]. |
| Name | The SyncSource name. [e.g.  “note” ] |
| Client Type | The preferred content type used talking with the Syncml clients (sif-n or text/plain) |

## 2.5. Officer Configuration

In order to set up the Officer for the Funambol JSON Connector, set the parameters in the file:

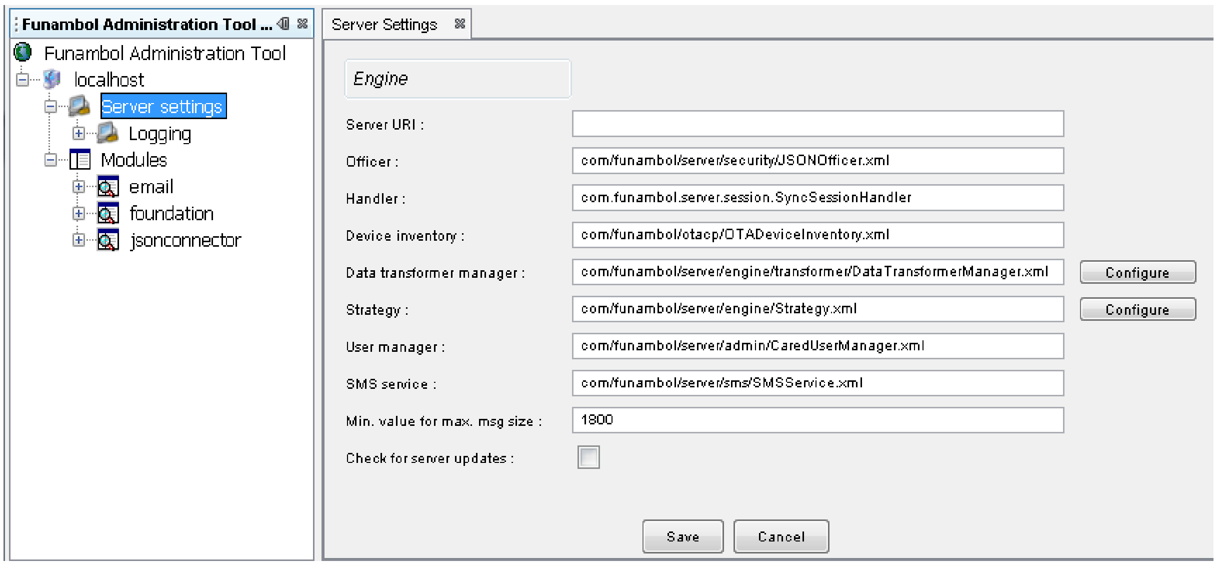
*$FUNAMBOL_HOME/ds-server/config/funambol/server/security/JsonOfficer.xml*

Below is an example of the *JsonOfficer.xml* file:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0" class="java.beans.XMLDecoder">

    <object class="com.funambol.json.security.JsonOfficer">

        <void property="serverAuth">
            <string>none</string>
        </void>

        <void property="autoProvisioning">
            <boolean>false</boolean>
        </void>

        ... device configuration for OTA ad PUSH env.

</java>
```

**Note:** if the *autoProvisioning* property is true, the officer will add it to the Funambol DS schema after authenticating the user on the back-end system.

In the Funambol Administration Tool, you must specify the appropriate Officer (as shown in Figure 3), that is: *com/funambol/server/security/JsonOfficer.xml.*

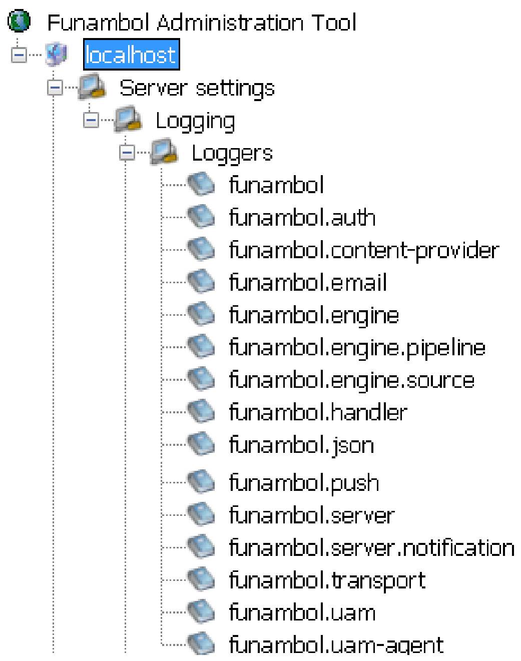


| | |
|---|---|
| Server URI : | |
| Officer : | com/funambol/server/security/JSONOfficer.xml |
| Handler : | com.funambol.server.session.SyncSessionHandler |
| Device inventory : | com/funambol/otacp/OTADeviceInventory.xml |
| Data transformer manager : | com/funambol/server/engine/transformer/DataTransformerManager.xml |
| Strategy : | com/funambol/server/engine/Strategy.xml |
| User manager : | com/funambol/server/admin/CaredUserManager.xml |
| SMS service : | com/funambol/server/sms/SMSService.xml |
| Min. value for max. msg size : | 1800 |
| Check for server updates : | ☐ |

*Figure 3: Officer Settings*

## 2.6. Enabling Logging

The logging level and other properties can also be configured using the Funambol Administration Tool, by expanding the tree structure shown in Figure 4:



```
Funambol Administration Tool
localhost
  Server settings
    Logging
      Loggers
        funambol
        funambol.auth
        funambol.content-provider
        funambol.email
        funambol.engine
        funambol.engine.pipeline
        funambol.engine.source
        funambol.handler
        funambol.json
        funambol.push
        funambol.server
        funambol.server.notification
        funambol.transport
        funambol.uam
        funambol.uam-agent
```

*Figure 4: Logging Tree*

To set the JSON Connector's logger, double click on the *funambol.json* node in the *Logging -> Loggers* tree (see Figure 4) and modify the options to the desired logging level and output.

## 2.7. Enabling Data Transformation

The Funambol server allows to configure encryption (DES, etc.) and encoding (BASE64, etc) parameters per sync source.

For legacy SIF data format, the BASE64 encoding must be enabled, for clients using standard formats (vCard/vCalendar), encoding is not required.

You can set the encoding method in the *Data Transformation* panel, found in the *Server Settings* section of the Funambol Administration Tool (see Figure 5).

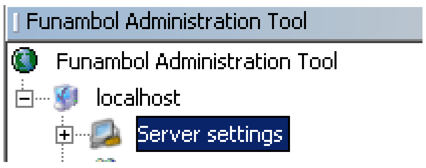
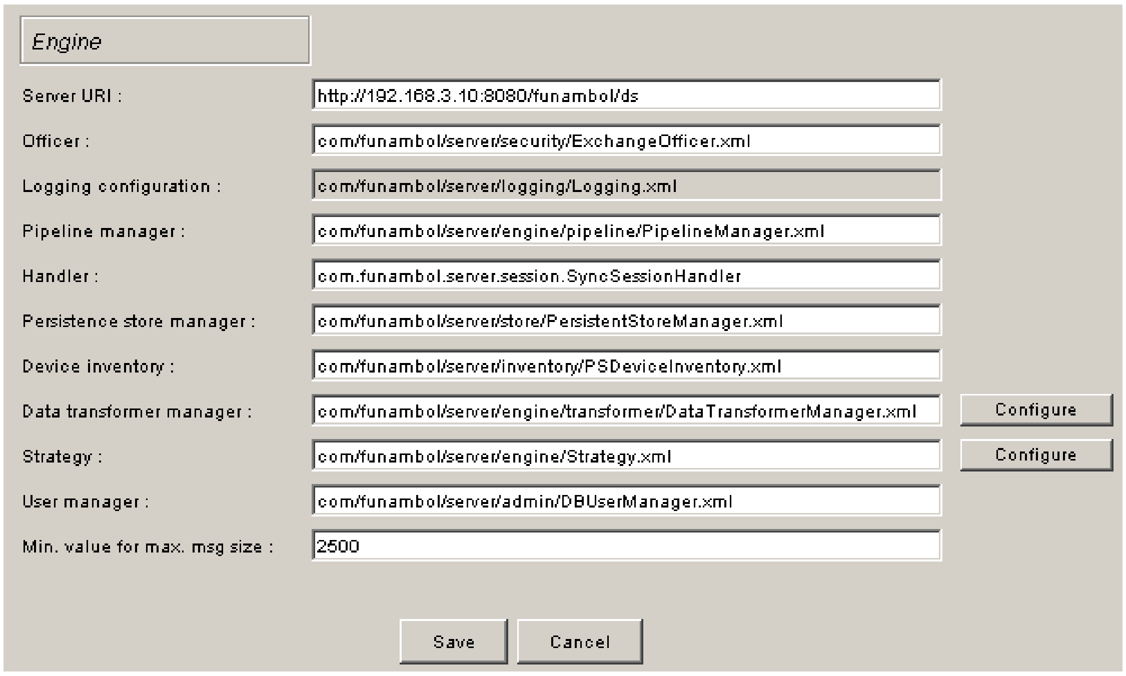


*Figure 5: Server Settings*

Expand the *Server settings* branch; in the configuration panel that appears on the right press the *Configure* button next to the *Data transformer manager* text field (see Figure 6).

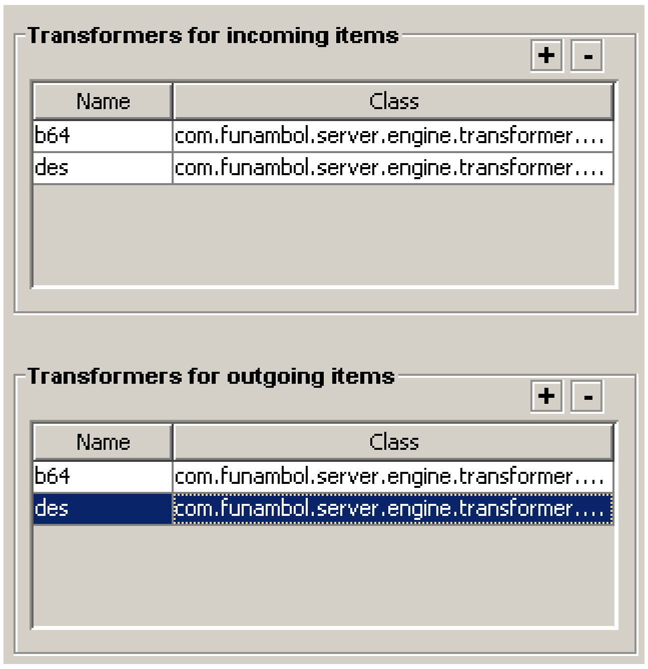


Engine

| | |
|---|---|
| Server URI : | http://192.168.3.10:8080/funambol/ds |
| Officer : | com/funambol/server/security/ExchangeOfficer.xml |
| Logging configuration : | com/funambol/server/logging/Logging.xml |
| Pipeline manager : | com/funambol/server/engine/pipeline/PipelineManager.xml |
| Handler : | com.funambol.server.session.SyncSessionHandler |
| Persistence store manager : | com/funambol/server/store/PersistentStoreManager.xml |
| Device inventory : | com/funambol/server/inventory/PSDeviceInventory.xml |
| Data transformer manager : | com/funambol/server/engine/transformer/DataTransformerManager.xml |
| Strategy : | com/funambol/server/engine/Strategy.xml |
| User manager : | com/funambol/server/admin/DBUserManager.xml |
| Min. value for max. msg size : | 2500 |

*Figure 6: Engine Settings*

If needed, modify *Transformer for incoming items* and *Transformer for outgoing items* (see Figure 7).


Transformers for incoming items
+
-
Transformers for outgoing items
+
-

| Name | Class |
|---|---|
| b64 | com.funambol.server.engine.transformer.... |
| des | com.funambol.server.engine.transformer.... |

| Name | Class |
|---|---|
| b64 | com.funambol.server.engine.transformer.... |
| des | com.funambol.server.engine.transformer.... |

*Figure 7: Transformers*

This information is needed when you have to configure SIF sync sources from the Admin tool (see next chapter).

# 3. Creating the  SyncSources

In this section you will find detailed instructions on how to create the SyncSources for the Json connector. By default, the connector does not create SyncSources, and you have two options:

- replace the default SyncSources, so you have card, event and task sync sources linked to Json Connector instead of the Foundation (this is the preferred choice, because the SyncML clients do not need to change the remote names):
- create new SyncSources with different names: in this case, the new name must be specified in the clients to use the Json connector. This is suggested ONLY if you really want to maintain both Foundation SyncSources and Json SyncSources on the same server, leaving the choice to the end user.

In the following chapters the steps to replace the default SyncSources are explained.

## 3.1. Steps to replace the defaut SyncSources using the command line.

To replace SyncSources without using the Funambol Administration Tool, you must modify the Funambol database, setting all the default sources in the table *fnbl_sync_source*  table to point to the JSON connector definitions.

You can verify the table content before and after the change the command:

```
select * from fnbl_sync_source;
```

To replace the default SyncSources, run the following queries:

```
update fnbl_sync_source set config='jsonconnector/jsonconnector/calendar-json/cal.xml' where
uri='cal';

update fnbl_sync_source set sourcetype='calendar-json' where uri='cal';

update fnbl_sync_source set config='jsonconnector/jsonconnector/contact-json/card.xml' where
uri='card';

update fnbl_sync_source set sourcetype='contact-json' where uri='card';

update fnbl_sync_source set config='jsonconnector/jsonconnector/calendar-json/event.xml' where
uri='event';

update fnbl_sync_source set sourcetype='calendar-json' where uri='event';

update fnbl_sync_source set config='jsonconnector/jsonconnector/note-json/note.xml' where
uri='note';

update fnbl_sync_source set sourcetype='note-json' where uri='note';

update fnbl_sync_source set config='jsonconnector/jsonconnector/calendar-json/scal.xml' where
uri='scal';

update fnbl_sync_source set sourcetype='calendar-json' where uri='scal';

update fnbl_sync_source set config='jsonconnector/jsonconnector/contact-json/scard.xml' where
uri='scard';

update fnbl_sync_source set sourcetype='contact-json' where uri='scard';

update fnbl_sync_source set config='jsonconnector/jsonconnector/note-json/snote.xml' where
uri='snote';

update fnbl_sync_source set sourcetype='note-json' where uri='snote';

update fnbl_sync_source set config='jsonconnector/jsonconnector/calendar-json/stask.xml' where
uri='stask';

update fnbl_sync_source set sourcetype='calendar-json' where uri='stask';
update fnbl_sync_source set config='jsonconnector/jsonconnector/calendar-json/task.xml' where
uri='task';

update fnbl_sync_source set sourcetype='calendar-json' where uri='task';
```

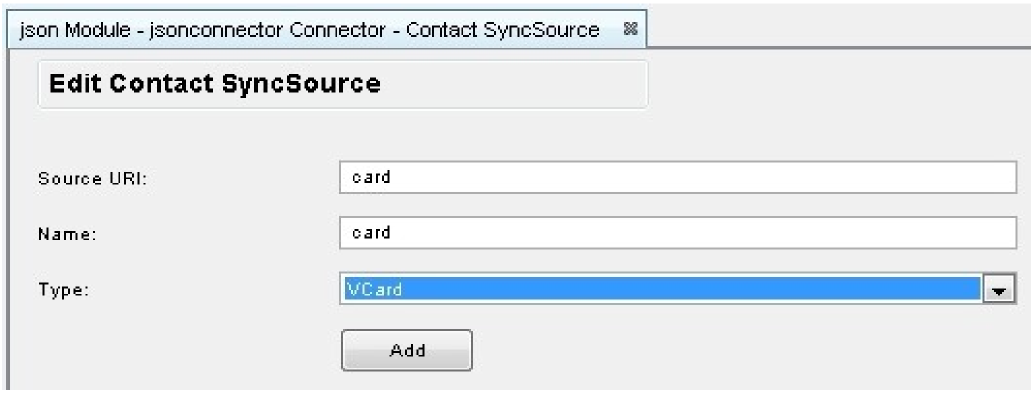
## 3.2. Steps to replace the standard SyncSources using the Admin tool.

### 3.2.1. Steps to replace card / scard

In order to replace *card* / *scard*, follow these steps:

1. Run the Funambol Administration Tool and connect to your local Funambol Server.
2. Select *Modules* -> *foundation* -> *FunambolFoundationConnector* -> *PIMContact Sync Source* -> *card*; right click on *card* and select “Delete” from the menu.
3. Select *Modules* -> *foundation* -> *FunambolFoundationConnector* -> *PIMContact Sync Source* -> *scard*; right click on *scard* and select “Delete” from the menu.
4. Select *Modules* -> *JsonConnector* -> *JsonConnector* -> *Contact SyncSource*; right click on it and select “Add Syncsource” from the menu.

   Edit the fields as shown in Figure 8.



json Module - jsonconnector Connector - Contact SyncSource

**Edit Contact SyncSource**

Source URI: card

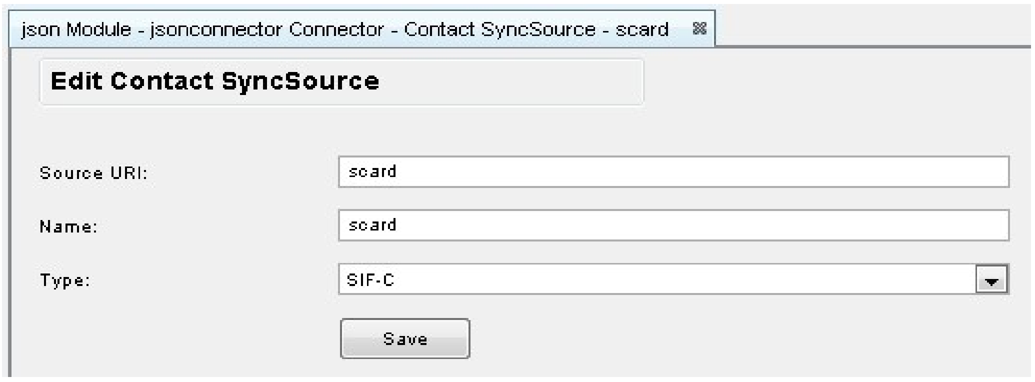
Name: card

Type: VCard

*Figure 8: Adding the card SyncSource*

5. Select *Modules* -> *JsonConnector* -> *JsonConnector* -> *Contact SyncSource*; right click on it and select “Add Syncsource” from the menu.

   Edit the fields as shown in Figure 9.



json Module - jsonconnector Connector - Contact SyncSource - scard

**Edit Contact SyncSource**

Source URI: scard

Name: scard

Type: SIF-C

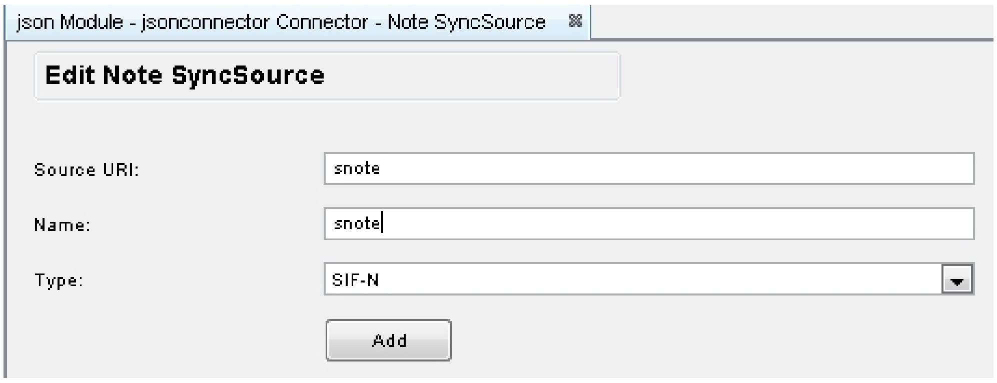
*Figure 9: Adding the scard SyncSource*

6. Select *Server Settings* from the menu and then press the *Configure* button in the *Data Transformation Manager* panel. Add the entry “scard, b64”.

### 3.2.2. Steps to replace note / snote

In order to replace *note* / *snote*, follow these steps:

1. Run the Funambol Administration Tool and connect to your local Funambol Server.
2. Select *Modules -> foundation -> FunambolFoundationConnector -> Notes SyncSource -> note*; right click on *note* and select “Delete” from the menu.
3. Select *Modules -> foundation -> FunambolFoundationConnector -> Notes SyncSource -> snote*; right click on *snote* and select “Delete” from the menu.
4. Select *Modules -> JsonConnector -> JsonConnector -> Note SyncSource*; right click on it and select  “Add Syncsource” from the menu.
5. Edit the fields as shown in Figure 11.


json Module - jsonconnector Connector - Note SyncSource

**Edit Note SyncSource**

Source URI: snote

Name: snote

Type: SIF-N

*Figure 10: Adding the snote SyncSource*

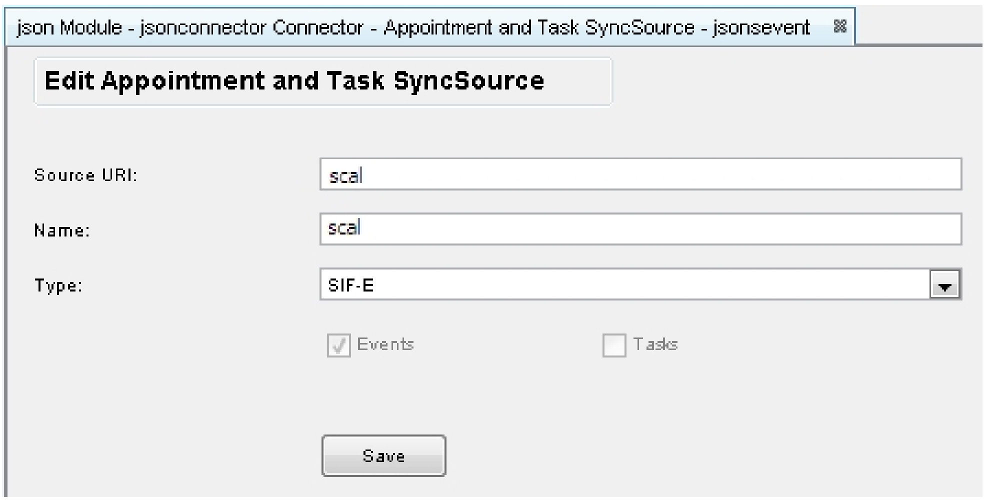
6. Select *Modules -> JsonConnector -> JsonConnector -> Note SyncSource*; right click on it and select  “Add Syncsource” from the menu. Edit the fields as shown in Figure 10.
7. Select *Server Settings* from the menu and then press the *Configure* button in the *Data Transformation Manager* panel. Add the entry “snote, b64”.

### 3.2.3. Steps to replace cal / task

In order to replace *cal* / *task*, follow these steps:

1. Run the Funambol Administration Tool and connect to your local Funambol Server.
2. Select *Modules -> foundation -> FunambolFoundationConnector -> PIM Calendar SyncSource*; delete *cal*, *scal*, *task*, *stask* and then *event* by right clicking on each item and selecting “Delete” from the menu.
3. Select *Modules -> JsonConnector -> JsonConnector -> Appointment and Task SyncSource*; right click on it and select  “Add Syncsource” from the menu.

   Edit the fields as shown in Figure 12.



json Module - jsonconnector Connector - Appointment and Task SyncSource - jsonsevent

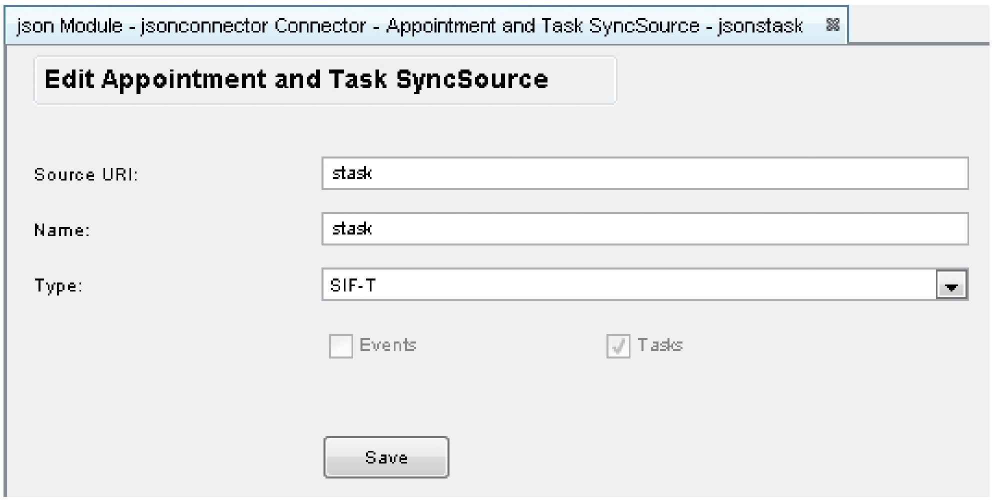
**Edit Appointment and Task SyncSource**

| | |
|---|---|
| Source URI: | scal |
| Name: | scal |
| Type: | SIF-E |
| [✓ Events] | Tasks |

*Figure 12: Adding the scal SyncSource*

4. Select *Modules* -> *JsonConnector* -> *JsonConnector* -> *Appointment and Task SyncSource*; right click on it and select "Add Syncsource" from the menu.

   Edit the fields as shown in Figure 13.



json Module - jsonconnector Connector - Appointment and Task SyncSource - jsonstask

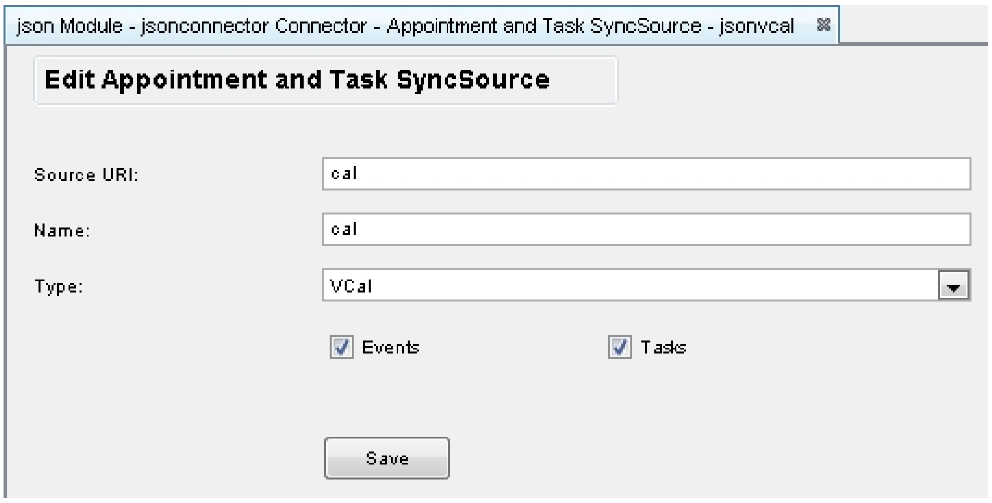
**Edit Appointment and Task SyncSource**

| | |
|---|---|
| Source URI: | stask |
| Name: | stask |
| Type: | SIF-T |
| Events | [✓ Tasks] |

*Figure 13: Adding the stask SyncSource*

5. Select *Modules* -> *JsonConnector* -> *JsonConnector* -> *Appointment and Task SyncSource*; right click on it and select "Add Syncsource".

   Edit the fields as shown in Figure 14.


json Module - jsonconnector Connector - Appointment and Task SyncSource - jsonvcal

**Edit Appointment and Task SyncSource**

Source URI: cal

Name: cal

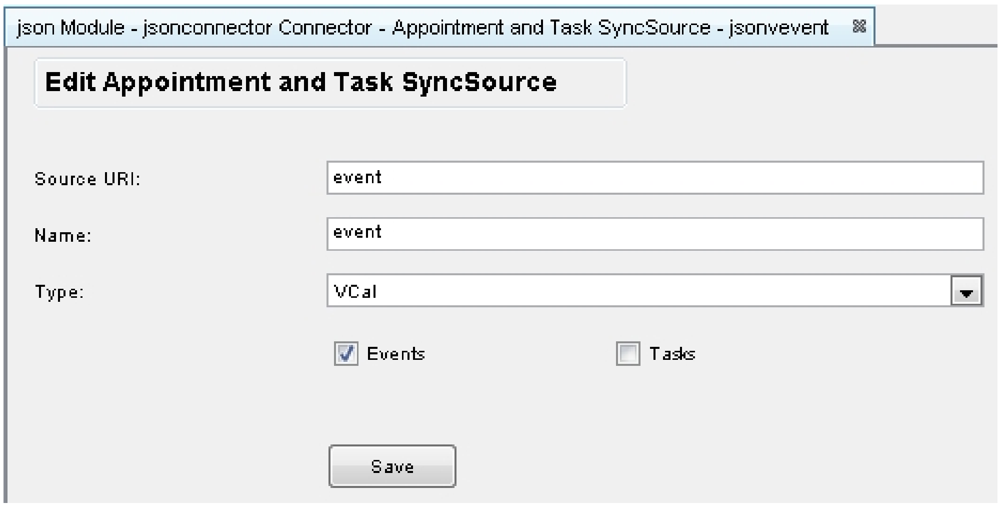
Type: VCal

☑ Events ☑ Tasks

*Figure 14: Adding the cal SyncSource*

6. Select *Modules -> JsonConnector -> JsonConnector -> Appointment and Task SyncSource*; right click on it and select "Add SyncSource".

   Edit the fields as shown in Figure 15.


json Module - jsonconnector Connector - Appointment and Task SyncSource - jsonvevent

**Edit Appointment and Task SyncSource**

Source URI: event

Name: event

Type: VCal

☑ Events ☐ Tasks

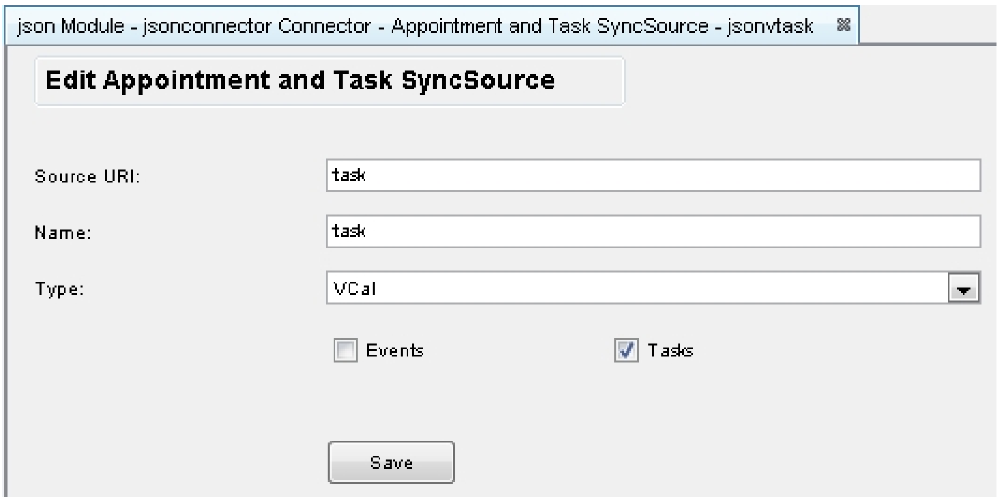
*Figure 15: Adding the event SyncSource*

**Note**: the *cal* SyncSource is used for devices that use the same SyncSource for tasks and events (such as Nokia phones). The *event* SyncSource should be used for devices that are able to synchronize tasks and events separately; in this case, the *task* SyncSource should be used to synchronize tasks.

7. Select *Modules -> JsonConnector -> JsonConnector -> Appointment and Task SyncSource*; right click on it and select "Add Syncsource".

   Edit the fields as shown in Figure 16.

8. Select *Server Settings* from the menu and then press the *Configure* button in the *Data Transformation Manager* panel. Add the following entries to the Data Transformations table: "scal, b64", "stask, b64".


json Module - jsonconnector Connector - Appointment and Task SyncSource - jsonvtask

Edit Appointment and Task SyncSource

Source URI: task

Name: task

Type: VCal

[ ] Events [✓ Tasks]

*Figure 16: Adding the task SyncSource*

# 4. Upgrading notes

If you already have the JSON connector installed and you want to install a new version, you have to care about some steps to make sure you will not loose any configuration.

The installation process copies the config files from the s4j to your system, to make sure you have the latest version of them. If you made changes to them, out of the most common configuration, you need to re-apply those changes.

If you configured your sync sources as described in 3, and you left the default parameters for all the rest, you can install the new module repeating the steps described in 2.2, answering 'yes' to the question about database initialization.

If you used different names, you need to make sure the config files are updated. The simplest way to do it is to remove the SyncSources and re-create them.

Other things you need to verify/restore:

1. the url of the datastore in jsonconnector.xml is restored to localhost:8080; if you changed it, you have to set it again;
2. the format used by the sync sources is defaulted to JSON Extended; if you are using a different configuration, you have to set it according to your backend; [**Note**: the configuration of the datastore format is changed in v8.7, because it is now per-source]

# 5. Solution architecture

The Funambol JSON Connector acts as a client connecting to the Funambol-specified JSON API implemented on the server side. This is used to provide an interface to an already existing data store to implement the Funambol synchronization service towards end users (mobile and desktop-based clients).

The data passed in the request and response of the REST [5] calls is encoded using JSON [4]. The connector can make use of either HTTP or HTTPS.

The Funambol JSON Connector translates Funambol SyncSource calls into REST calls using JSON to represent data and data structures.

## 5.1. JSON API protocol functionality

The JSON protocol provides the methods to authenticate user credentials with the endpoint of the external PIM data source.

The following functions are required for an Officer implementation:

- *authenticateUser (login)*: authenticates user credentials from the device with the server
- *authenticateUser (logout)*: logs out a previously authenticated user

The JSON protocol provides the required methods to retrieve, update and search the data stored in the endpoint of the external PIM data source.

The following functions are required for a SyncSource implementation:

- *beginSync*: tells the server that a sync session is going to begin
- *endSync*: tells the server that a sync session has ended
- *getAllSyncItemKeys*: gets all the keys for the items in the server
- *getDeletedSyncItemKeys*: gets all the keys of the deleted items since a given time
- *getUpdatedSyncItemKeys*: gets all the keys of the updated items since a given time
- *getNewSyncItemKeys*: gets all the keys of the new items since a given time
- *getSyncItemFromId*: gets the item associated with a given key
- *removeSyncItem*: removes the item with a given key
- *addSyncItem*: inserts an item in the server and returns it with a key
- *updateSyncItem*: updates an item in the server for a specific key
- *getSyncItemKeysFromTwin*: searches for application-defined equivalent items for a given item

**Note:** developers will implement the JSON API as an extension of their existing application.

## 5.2. Appointment, Contact, Note and Task SyncSources

The Appointment, Contact, Note and Task SyncSources provide access to the PIM information of the backend (the JSON server).

The backend implementation can choose, for each SyncSource, to represent the items using standard vCard or vCalendar objects embedded into the Json call, or to use a Json structure to represent also the item (also referred as JSON extended format).

The advantage of the Json Extended format is to provide a simpler format to handle: if you are developing your datastore and you do not have a vCard/vCalendar parser available for your environment, dealing with the JSON representation is simpler.

If you already have the capability of handling vCard/vCalendar, using this format can give you the possibility to support more information. The JSON connector, anyway, is tested only with the fields currently supported by the Funambol Foundation connector. Going beyond this is possible, but may require addional testing.

When you install or upgrade the connector, the format used by default is JSON Extended.

Summarizing, the current version of the specification use the following formats to represent PIM data:

- Contacts:
  - extended JSON structure
  - JSON structure with vCard
- Appointment (or Calendar):
  - extended JSON structure
  - JSON structure with vCalendar / iCalendar (vEvent)
- Task:
  - extended JSON structure (same as appointment)
  - JSON structure with vCalendar / iCalendar (vTodo)
- Note:
  - extended JSON structure

For the JSON structure specification, see [6]. For vCard and vCalendar / iCalendar see [7] and [8] respectively.

When synchronizing notes, the JSON Connector will receive SIF-N (Funambol format) and UTF-8 plain/text format but it will always send an extended JSON structure to the JSON API layer. At the moment, the vNote format is not supported.

The JSON Connector is responsible for converting the above formats into the appropriate format for the mobile device, which is determined by configuration. Each format will have its own URI to address the SyncSource. In particular, the JSON Connector converts vCalendar 2.0 (iCalendar) format coming from the server / database into vCalendar 1.0, which is supported by most clients, or SIF, for backward compatibility with old Funambol clients.

### 5.2.1. A quick review of vCalendar / iCalendar

| ***vCalendar 1.0*** | ***vCalendar 2.0 (also called iCalendar)*** |
|---|---|
| content:<br>BEGIN:VCALENDAR<br>VERSION:1.0<br>BEGIN:VEVENT<br>...<br>END:VEVENT<br>END:VCALENDAR<br>MIME type:<br>text/x-vcalendar | content:<br>BEGIN:VCALENDAR<br>VERSION:2.0<br>BEGIN:VEVENT<br>...<br>END:VEVENT<br>END:VCALENDAR<br>MIME type:<br>text/calendar |

The main differences between these two formats can be found in:

- RRULE
- ALARM
- TIMEZONE

## 5.3. JSON API specification overview

This section gives a definition of the API grouped by functionality.

The following API groups are defined:

- **auth**: used to authenticate the caller and obtain a session identifier to use in subsequent calls
- **sync**: used to notify the begin and end of a synchronization session
- **keys**: used to get all, new, updated, deleted and twin keys from the SyncSource
- **items**: used to add, update, remove and get items from the SyncSource
- **utilities**: miscellaneous utility APIs

## 5.4. Conventions

The description of each API group mentioned above, that you can find later in this document, includes a list of all the available actions. Actions are described with:

- the HTTP request (method and URI) to be invoked
- the request parameters
- the expected response
- errors that the request could return
- examples

### 5.4.1. Request

The pattern of the URL for a request is */sync-source-name/group/action*, where:

- *sync-source-name* is the name of the SyncSource; for example:
    - appointment
    - contact
    - task
    - note
- *group* is the API group (one of: auth, sync, keys, items, utilities, as defined in 5.3)
- *action* is the action to be performed in the group

For example:

- the URL */contact/keys/all* would get all of the item keys for the contact SyncSource
- the URL */appointment/keys/all* would get all of the item keys for the appointment SyncSource

Any call that is required to send an object to the server must send a JSON object that contains the following property:

- **data**: the request payload

**Note**: encapsulating the request payload in a data property allows more flexibility for any future changes.

### 5.4.2. Response

All calls return a JSON object that can contain the following properties:

- **data**: the response payload

- **error**: error details, if any; the error payload contains an error code, an error message and a list of parameters

Below is an example of response containing the payload:

```
{"data":{
       "keys":[
              "key-1",
              "key-2",
              ...
              "key-n",
       ]
    }
}
```

Below is an example of response containing an error:

```
{"error":{
   "code":"ITEM-1004",
   "message":"Error item not found",
   "parameters":[
       {
              "param":"server-item-key"
       }
    ]
  }
}
```

If the call contains an incorrect request, the return must be a JSON object with the property: **error**.

Below is an example of an unsupported request error:

```
{"error":{
   "code":"ITEM-1005",
   "message":"Request is not yet supported-recognized.",
   "parameters":[]
  }
}
```

## 5.5. HTTP methods

Resources are accessed and modified by using standard HTTP methods. The Funambol REST architecture supports the following methods:

- **GET**: gets a resource without changing its actual state
- **POST**: creates a new resource
- **PUT**: updates the state of an existing resource
- **DELETE**: deletes an existing resource

## 5.6. HTTP status codes

Standard HTTP status codes should be used by the server to inform the clients of an operation result.

The Funambol REST architecture supports the following status codes:

| ***HTTP status code*** | ***HTTP status message*** | ***Description*** |
|---|---|---|
| 200 | OK | Informs the client that a given operation has been successfully completed. |
| 401 | Unauthorized | Informs clients that a given operation requires the client to be authenticated. |
| 406 | Not acceptable | Informs clients that a given operation failed due to errors in the client request. |
| 500 | Internal server error | Informs clients that a given operation failed due to internal problems in the server application (more information should be contained into the server response). |

# 6. Data object format

## 6.1. Contact object format – JSON extended format

The *contact* resource in JSON extended format is described by the following JSON schema:

```
{ "description" : "Contact resource", "type" : "object", "properties" : {
       "data"  :   {  "description"  :  "Contact  resource  data",  "type"  :  "object",
"properties" : {
        "content-type" : { "description" : "Resource content-type",
                           "type" : "string",
                           "default" : "application/json-card"},
        "item" : { "description" : "Resource data", "type" : "object", "properties" : {
            "key" : {"description" : "Resource identifier", "type" : "string"},
            "state" : {"description" : "Resource state", "type" : "string"},
            "lastUpdate" : {"description" : "Resource update time", "type" : "string"},
            "folder" : {"type" : "string"},
            "title" : {"type" : "string"},
            "firstName" : {"type" : "string"},
            "middleName" : {"type" : "string"},
            "lastName" : {"type" : "string"},
            "suffix" : {"type" : "string"},
            "photo" : { "description" : "Base64-encoded picture", "type" : "string" },
            "photoType" : {"description" : "Picture format", "type" : "string"},

            "email" : {"type" : "string"},
            "email2" : {"type" : "string"},
            "email3" : {"type" : "string"},
            "imAddress" : {"type" : "string"},

            "businessAddressStreet" : {"type" : "string"},
            "businessAddressCity" : {"type" : "string"},
            "businessAddressCountry" : {"type" : "string"},
            "businessAddressState" : {"type" : "string"},
            "businessAddressPostalCode" : {"type" : "string"},
            "businessAddressPostOfficeBox" : {"type" : "string"},
            "businessAddressExtendedAddress" : {"type" : "string"},

            "homeAddressStreet" : {"type" : "string"},
            "homeAddressCity" : {"type" : "string"},
            "homeAddressState" : {"type" : "string"},
            "homeAddressCountry" : {"type" : "string"},
            "homeAddressPostalCode" : {"type" : "string"},
            "homeAddressPostOfficeBox" : {"type" : "string"},
            "homeAddressExtendedAddress" : {"type" : "string"},
            "otherAddressStreet" : {"type" : "string"},
            "otherAddressCity" : {"type" : "string"},
            "otherAddressState" : {"type" : "string"},
            "otherAddressCountry" : {"type" : "string"},
            "otherAddressPostalCode" : {"type" : "string"},
            "otherAddressPostOfficeBox" : {"type" : "string"},
            "otherAddressExtendedAddress" : {"type" : "string"},

            "phoneAssistant" : {"type" : "string"},
            "phoneBusiness" : {"type" : "string"},
            "phoneBusiness2" : {"type" : "string"},
            "phoneBusinessFAX" : {"type" : "string"},
            "phoneCallback" : {"type" : "string"},
            "phoneCar" : {"type" : "string"},
            "phoneCompany" : {"type" : "string"},
            "phoneHome" : {"type" : "string"},
            "phoneHome2" : {"type" : "string"},
            "phoneHomeFAX" : {"type" : "string"},
            "phoneMobile" : {"type" : "string"},
            "phoneMobileHome" : {"type" : "string"},
            "phoneMobileBusiness" : {"type" : "string"},
            "phoneOther" : {"type" : "string"},
            "phoneOtherFAX" : {"type" : "string"},
            "phonePrimary" : {"type" : "string"},
            "phonePager" : {"type" : "string"},
            "phoneRadio" : {"type" : "string"},
            "phoneTelex" : {"type" : "string"},
            "url" : {"type" : "string"},
            "instantMessenger1" : {"type" : "string"},
            "body" : {"type" : "string"},
            "jobTitle" : {"type" : "string"},
            "company" : {"type" : "string"},
            "sensitivity" : {"type" : "int"},
            "department" : {"type" : "string"},
            "office" : {"type" : "string"},
            "managerName" : {"type" : "string"},
            "assistantName" : {"type" : "string"},
            "nickName" : {"type" : "string"},
            "spouseName" : {"type" : "string"},
            "anniversary" : {"type" : "string", "description" : "format YYYY-MM-DD"},
            "birthday" : {"type" : "string", "description" : "format YYYY-MM-DD"}
        }}
    }}
}}
```

Below is an example of a JSON data structure containing contact information:

```
{ "data" : {
        "content-type" : "application/json-card",
        "item" : {
                "key": "12ADE43",
                "folder": "root\Contacts",
                "firstName": "John",
                "lastName": "Smith",
                "photo": "/9j/4AAQSkZJRgABAgAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKD",
                "photoType": "GIF",
                "email": "john.smith@email.ti",
                "company": "maruzzella srl",
                ...
                "anniversary": "2005-06-17",
                "birthday": "1970-12-12"
         }
}}
```

For more examples, see Error: Reference source not found.

## 6.2. Contact object format – vCard format

The *contact* resource in vCard format is described by the following JSON schema:

```
{ "description" : "Contact resource", "type" : "object", "properties" : {
        "data"  :  {  "description"  :  "Contact  resource  data",  "type"  :  "object",
"properties" : {
        "content-type" : { "description" : "Resource content-type",
                           "type" : "string",
                           "default" : "application/json-vcard"},
        "item" : { "description" : "Resource data", "type" : "object", "properties" : {
            "key" : {"description" : "Resource identifier", "type" : "string"},
            "state" : {"description" : "Resource state", "type" : "string"},
            "lastUpdate" : {"description" : "Resource update time", "type" : "string"},
            "folder" : {"type" : "string"},
            "vcard" : {"description" : "VCard RFC specs", "type" : "string"}
        }}
    }}
}}
```

Below is an example of a JSON data structure containing contact information:

```
{ "data" : {
        "content-type" : "application/json-vcard",
        "item" : {
                "key": "12ADE43",
                "vcard": "BEGIN:VCARD
```

```
              VERSION:2.1
              REV:20071031T222710
              UID:20070401-080045-40000F192713-0052
              CLASS:PUBLIC
              X-TOKEN:VCard 2.1
              N:Public;Tracy;Spencer;Mrs.;Esq.
              FN:Mrs. Tracy S. Public, Esq.
              EMAIL;INTERNET;HOME:tracy@tracy.org
              URL;HOME:http://www.tracy.org
              ORG:International Import/Export, Inc.;North American Division;Marketing
              ADR;WORK:;;1200 A1 California;Pasadena;CA;91125;USA
              ADR;DOM;HOME:P.O. Box 101;Suite 101;123 Main Street;Any Town;CA;91921-
1234;
              TEL;WORK:12358769712
              TEL;HOME:+1-919-555-1234
              TEL;WORK;HOME;VOICE;FAX:+1-800-555-1234
              URL:http://abc.com/pub/directory/northam/jpublic.ecd
              URL;WORK:http://www.impexp.com
              MAILER:ccMail 2.2
              TZ:-0500
              GEO:37.24,-17.87
              NOTE;ENCODING=QUOTED-PRINTABLE:Don't forget to order Girl Scout cookies
from Stacey today!
              PHOTO;VALUE=URL;TYPE=GIF:http://www.abc.com/dir_photos/my_photo.gif
              SOUND;VALUE=CONTENT-
ID:&lt;jsmith.part3.960817T083000.xyzMail@host1.com&gt;
              BDAY:19950415
              TITLE:V.P., Research and Development
              ROLE:Executive
              END:VCARD"
        }
}}
```

## 6.3. Appointment object format – JSON extended format

The *appointment* resource in JSON extended format is described by the following JSON schema:

```
{ "description" : "Appointment resource", "type" : "object", "properties" : {
      "data" : { "description" : "Appointment resource data", "type" : "object",
"properties" : {
        "content-type" : { "description" : "Resource content-type",
                           "type" : "string",
                           "default" : "application/json-appointment"},
        "item" : { "description" : "Resource data", "type" : "object", "properties" : {
            "key" : {"description" : "Resource identifier", "type" : "string"},
            "state" : {"description" : "Resource state", "type" : "string"},
            "lastUpdate" : {"description" : "Resource update time", "type" : "string"},
            "folder" : {"type" : "string"},
            "subject" : {"type" : "string"},
            "body" : {"type" : "string"},
            "location" : {"type" : "string"},
            "allDay" : {"type" : "boolean"},
            "startDate" : {"type" : "string" , "description" :
                           "YYYYMMDD format if all day,
                            YYYYMMDD'T'hhmmss if tzid is specified,
                            YYYYMMDD'T'hhmmss'Z' format otherwise"},
            "endDate" : {"type" : "string", "description" :
                           "YYYYMMDD format if all day,
                            YYYYMMDD'T'hhmmss if tzid is specified,
                            YYYYMMDD'T'hhmmss'Z' format otherwise"},
            "tzid" : {"type" : "string"},
            "reminder" : {"type" : "int"},
            "reminderTime" : {"type" : "string"},
            "busyStatus" : {"type" : "string"},
            "categories" : {"type" : "string"},
            "sensitivity" : {"type" : "string"},
            "importance" : {"type" : "string"},
            "isRecurring" : {"type" : "boolean"},
            "recurrenceType" : {"type" : "string"},
            "dayOfMonth" : {"type" : "string"},
            "dayOfWeekMask" : {"type" : "string"},
            "instance" : {"type" : "string"},
            "interval" : {"type" : "string"},
            "monthOfYear" : {"type" : "string"},
            "patternStartDate" : {"type" : "string", "description" :
                              "YYYYMMDD format if all day,
                              YYYYMMDD'T'hhmmss format otherwise"},
            "occurrences" : {"type" : "string"},
            "noEndDate" : {"type" : "boolean"},
            "patternEndDate" : {"type" : "string", "description" :
                              "YYYYMMDD format if all day,
                              YYYYMMDD'T'hhmmss format otherwise"},
            "exceptionsExcluded" : {"type" : "array"},
            "exceptionsIncluded" : {"type" : "array"}
        }}
    }}
}}
```

Below is an example of a JSON data structure containing appointment information (single appointment):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "holiday",
       "tzid": "America/Mexico_City",
       "startDate": "20081001T170000Z",
       "endDate": "20081001T180000Z",
       "location": "Acapulco",
       ...
       "allDay": false,
       "isRecurring": false
}
}}
```

Below is an example of a JSON data structure containing appointment information (single all-day appointment):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "holiday all day",
        "tzid": "America/Mexico_City",
        "startDate": "20081001",
        "endDate": "20081001",
        "location": "Acapulco",
        ...
        "allDay": true,
        "isRecurring": false
 }
}}
```

For more examples see Error: Reference source not found.

## 6.4. Appointment object format – vCalendar / iCalendar format

The appointment resource in vCalendar / iCalendar format is described by the following JSON schema:

```
{ "description" : "Appointment resource", "type" : "object", "properties" : {
       "data"  :  {  "description"  :  "Appointment  resource  data",  "type"  :  "object",
"properties" : {
        "content-type" : { "description" : "Resource content-type",
                           "type" : "string",
                           "default" : "application/json-vcal"},
        "item" : { "description" : "Resource data", "type" : "object", "properties" : {
            "key" : {"description" : "Resource identifier", "type" : "string"},
            "state" : {"description" : "Resource state", "type" : "string"},
            "lastUpdate" : {"description" : "Resource update time", "type" : "string"},
            "folder" : {"type" : "string"},
                 "vcal"  :  {"description"  :  "VCalvCalendarendar  RFC  specs",  "type"  :
"string"}
        }}
    }}
}}
```

Below is an example of a JSON data structure containing vCalendar information:

```
{ "data" : {
        "content-type" : "application/json-vcal",
        "item" : {
                "key": "12ADE43",
                "vcal": "BEGIN:VCALENDAR
                PRODID:-//ABC Corporation//NONSGML My Product//EN
                TZ:+05:30
                VERSION:1.0
                BEGIN:VEVENT
                AALARM;TYPE=WAVE;VALUE=URL:20070415T235900; ; ; file:///mmedia/taps.wav
                DALARM:20070415T235000;PT5M;2;Your Taxes Are Due !!!
                PALARM;VALUE=URL:20070415T235000;PT5M;2;file:///myapps/shockme.exe
                MALARM:20070416T000000;PT1H;24;IRS@us.gov;The Check Is In The Mail!
                CATEGORIES:APPOINTMENT;EDUCATION
                CLASS:PUBLIC
                LAST-MODIFIED:20070817T133000Z
                DCREATED:20070329T083000
                COMPLETED:20070401T235900
                DESCRIPTION;ENCODING=QUOTED-PRINTABLE:Don’t forget to order Girl Scout
cookies from Stacey today!
                EXDATE:20070402T010000Z;20070403T010000Z;20070404T010000Z
                XRULE:D1 #10
                DUE:20070401T235900Z
                LOCATION;VALUE=URL;TYPE=VCARD:http://www.xyzcorp.com/~jsmith.vcf
                RNUM:3
                PRIORITY:2
                RELATED-TO:&lt;jsmith.part7.19960817T083000.xyzMail@host3.com&gt;
                RDATE:20070402T010000Z;20070403T010000Z;20070404T010000Z
                RRULE:W5 TU TH
                RESOURCES:EASEL;PROJECTOR;VCR
                SEQUENCE:1
                STATUS:TENTATIVE
                SUMMARY:Proposal Review
                TRANSP:0
                URL:http://abc.com/pub/calendars/jsmith/mytime.or3
                UID:20070401-080045-4000F192713-0052
                GEO:37.24,-17.87
                DAYLIGHT:TRUE;-06;20070407T025959;20071027T010000;EST;EDT
                DTSTART:20070614T150000Z
                DTEND:20070614T160000Z
                END:VEVENT
                END:VCALENDAR
         }
}}
```

Below is an example of a JSON data structure containing iCalendar information:

```
{ "data" : {
        "content-type" : "application/json-vcal",
        "item" : {
                "key": "12ADE43",
                "vcal": "BEGIN:VCALENDAR
                VERSION:2.0
                METHOD:PUBLISH
                BEGIN:VEVENT
                UID:20060416T204136Z-4272-727-1-247@gollum
                DTSTAMP:20060416T204136Z
                DTSTART:20060406T190000Z
                DTEND:20060406T193000Z
                TRANSP:TRANSPARENT
                SEQUENCE:4
                SUMMARY:all fields
                LOCATION:virtual
                CATEGORIES:Business,test,bar
                CLASS:PRIVATE
                CREATED:20080809T193645
                LAST-MODIFIED:20080809T193645
                DESCRIPTION:this is an appointment
                END:VEVENT
                END:VCALENDAR
         }
}}
```

Below is an example of a JSON data structure containing iCalendar information:

```
{ "data" : {
        "content-type" : "application/json-vcal",
        "item" : {
                "key": "12ADE43",
                "vcal": "BEGIN:VCALENDAR
                VERSION:2.0
                BEGIN:VTIMEZONE
                TZID:/mozilla.org/20050126_1/Europe/Berlin
                X-LIC-LOCATION:Europe/Berlin
                BEGIN:DAYLIGHT
                TZOFFSETFROM:+0100
                TZOFFSETTO:+0200
                TZNAME:CEST
                DTSTART:19700329T020000
                RRULE:FREQ=YEARLY;INTERVAL=1;BYDAY=-1SU;BYMONTH=3
                END:DAYLIGHT
```

```
              BEGIN:STANDARD
              TZOFFSETFROM:+0200
              TZOFFSETTO:+0100
              TZNAME:CET
              DTSTART:19701025T030000
              RRULE:FREQ=YEARLY;INTERVAL=1;BYDAY=-1SU;BYMONTH=10
              END:STANDARD
              END:VTIMEZONE
              BEGIN:VEVENT
              DTSTART;TZID=/mozilla.org/20050126_1/Europe/Berlin:20070929T140000
              DTEND:20070929T130000Z
              SUMMARY:Summary
              LOCATION:Location
              DESCRIPTION:Description
              END:VEVENT
              END:VCALENDAR
        }
}}
```

## 6.5. Task object format – JSON extended format

The task resource in JSON extended format is described by the following JSON schema:

```
{ "description" : "Task resource", "type" : "object", "properties" : {
  "data" : { "description" : " Task resource data", "type" : "object", "properties" : {
  "content-type" : { "description" : "Resource content-type",
                     "type" : "string" ,
                     "default" : "application/json-task"},
  "item" : { "description" : "Resource data", "type" : "object", "properties" : {
  "key" : {"description" : "Resource identifier", "type" : "string"},
  "state" : {"description" : "Resource synchronization state", "type" : "string"},
  "lastUpdate" : {"description" : "Resource update time", "type" : "string"},
  "folder" : {"type" : "string"},
  "subject" : {"type" : "string"},
  "body" : {"type" : "string"},
  "allDay" : {"type" : "boolean"},
  "startDate" : {"type" : "string" , "description" :
                                   "format YYYYMMDD if all day,
                                           YYYYMMDD'T'hhmmss if tzid is specified,
                                           YYYYMMDD'T'hhmmss'Z' otherwise"},
  "dueDate" : {"type" : "string", "description" :
                                   "format YYYYMMDD if all day,
                                           YYYYMMDD'T'hhmmss if tzid is specified,
                                           YYYYMMDD'T'hhmmss'Z' otherwise"},
  "tzid" : {"type" : "string"},
  "sensitivity" : {"type" : "string"},
  "importance" : {"type" : "string"},
  "status" : {"type" : "string"},
  "reminder" : {"type" : "boolean"},
  "reminderDate" : {"type" : "string", "description" :
                                   "format YYYYMMDD if all day,
                                           YYYYMMDD'T'hhmmss if tzid is specified,
                                           YYYYMMDD'T'hhmmss'Z' otherwise"},
  "complete" : {"type" : "boolean"},
  "percentComplete" : {"type" : "string"},
  "dateCompleted" : {"type" : "string"},
  "actualWork" : {"type" : "string"},
  "totalWork" : {"type" : "string"},
  "billingInformation" : {"type" : "string"},
  "companies" : {"type" : "string"},
  "mileage" : {"type" : "string"}
        }}
    }}
}}
```

## 6.6. Task object format – vCalendar / iCalendar format

In vCalendar / iCalendar a task has the same format as an appointment. Please see 6.4 Appointment object format – vCalendar / iCalendar format.

## 6.7. Notes object format

Information about note resources is described by the following JSON schema:

```
{ "description" : "Note resource", "type" : "object", "properties" : {
  "data" : { "description" : "Note resource data", "type" : "object", "properties" : {
  "content-type" : { "description" : "Resource content-type",
                     "type" : "string",
                     "default" : "application/json-note"},
  "item" : { "description" : "Resource data", "type" : "object", "properties" : {
  "key" : {"description" : "Resource identifier", "type" : "string"},
  "state" : {"description" : "Resource synchronization state", "type" : "string"},
            "folder" : {"type" : "string"},
            "subject" : {"type" : "string"},
            "body" : {"type" : "string"},
        }}
    }}
}}
```

Below is an example of a JSON data structure containing note information:

```
{ "data" : {
       "content-type" : "application/json-note",
       "item" : {
              "subject": "How to buy a puppets",
              "body": "go out and buy a puppets"
        }
}}
```

# 7. JSON Connector API

## 7.1. Authentication group

### 7.1.1. login

Authenticate user credentials with the SyncSource.

***Request***
POST /auth/login.

***Parameters***
None.

***Body***
The body of this request contains the following parameters:

- type
- user
- pass

***Response***
A string containing the session ID to send in all sync, keys, and items API requests.

***Error***
If there is an error authenticating the user, a 406 HTTP error code is returned and the content of the response is a JSON error. The following errors are expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_UNKNOWN_USER | The given credentials do not identify any existing user. | {"error":{ "code":"ERR_UNKNOWN_USER", "message":"Unknown user" } } |
| ERR_ACCOUNT_EXPIRED | The given credentials identify a user account that has expired; the parameter expiration-date specifies the expiration date. | {"error":{ "code":"ERR_ACCOUNT_EXPIRED", "message":"Account expired", "parameters":[{ "expiration-date":"20080808" }] } } |
| ERR_PAYMENT_REQUIRED | The user does not have enough credit. | {"error":{ "code":"ERR_PAYMENT_REQUIRED", "message":"Unknown user" } } |

***Example***
Request:

POST /auth/login.

Request body:

```
{"data":{
    "credentials":{
          "type": "basic",
          "user": "user1",
          "pass": "pass1"
       }
    }
}
```

Response:

```
{"data":{
        "sessionid":"4B339C8F5437B7A9506D8C69901833BF"
     }
 }
```

**Note:** username and password are sent as a string; this means that the credentials from a SyncML client are copied "as is" in the body of the Request.

**Note**: a previous implementation of /auth/login, accepting only 2 parameters (*user* and *password*), is now deprecated.

### 7.1.2. logout

Indicates that a previously login session can be discarded.

#### *Definition*

POST /auth/logout.

#### *Request header*

- **authorization**: the session ID returned in a previous login call

#### *Parameters*

None.

#### *Body*

- **sessionid**: the session ID returned in a previous login call

#### *Response*

An empty response.

#### *Errors*

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_INVALID_SESSION | The given session is invalid or unknown; the parameter sessionid specifies the offending session. | {"error":{ "code":"ERR_INVALID_SESSION", "message":"Invalid session", "parameters":[{ "sessionid":"A45568HD094" }] } } } |

***Example***

Request:

POST /auth/logout.

Request body:

```
{"data":{
    "sessionid":"4B339C8F5437B7A9506D8C69901833BF"
    }
}
```

## 7.2. Sync group

### 7.2.1. beginSync

Notify the JSON API that a sync is going to begin.

***Definition***

POST /<sync-source-name>/sync/begin.

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

- **synctype**: can be one of the following:
  - TWO-WAY: two-way synchronization
  - SLOW SYNC: full synchronization
  - ONE-WAY FROM SERVER: one-way from server synchronization
  - REFRESH FROM SERVER;  refresh from server synchronization
  - ONE-WAY FROM CLIENT: one-way from client synchronization
  - REFRESH FROM CLIENT: refresh from client synchronization
- **since**: the time of the start of the sync process in milliseconds (Unix timestamp in milliseconds)

***Response***

An empty response.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| *Error code* | *Description* | *Error object* |
|---|---|---|
| ERR_UNSUPPORTED_SYNC_TYPE | The given sync type is not supported; the parameter sync-type specifies the unsupported type. | {"error":{ "code":"ERR_UNSOPORTED_SYNC_TYPE", "message":"Unsupported sync type", "parameters":[{ "sync-type":"refresh-from-server" }] } } |

### *Example*

Request:

POST /contact/sync/begin.

Request body:

```
{"data":{
    "synctype":"two-way",
    "since":1222956766865
    }
}
```

### *The since parameter in the beginSync request*

This paragraph explains why the *beginSync* request sets the *since* parameter.

This parameter is related to the following methods:

- *getNewSyncItemKeys*
- *getUpdatedSyncItemKeys*
- *getDeletedSyncItemKeys*
- *addSyncItem*
- *updateSyncItem*
- *removeSyncItem*

The synchronization process is atomic. Figure 17 shows the scenario that involves *getNewSyncItemKeys*, *getUpdatedSyncItemKeys*, *getDeletedSyncItemKeys* (briefly *get_N_U_D*).

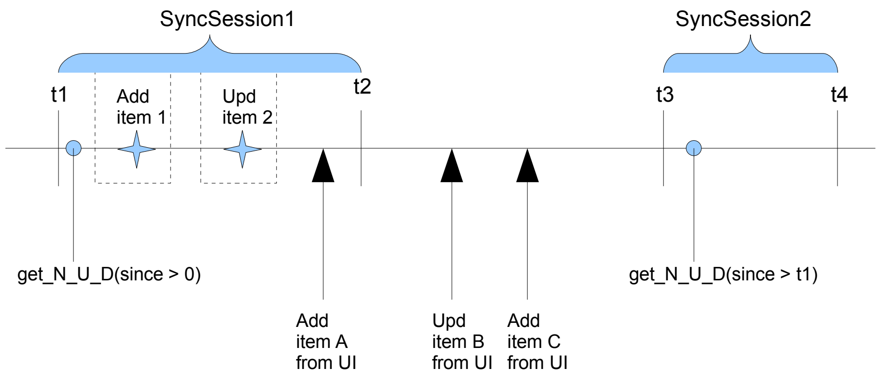


*Figure 17: The synchronization process*

The first synchronization process (i.e. *SyncSession1*) starts at time *t1* and ends at time *t2*; during this process, the system adds/updates items 1 and 2 (see Figure 17).

Items 1 and 2 must have the creation/update time equal to *t1* in the storage of the backend.

The add, update and delete operations are performed by the API, using the methods *addSyncItem*, *updateSyncItem* and *removeSyncItem*, so that the *t1* value must be taken into account and set as the creation/update time in the backend storage system.

This step is very important for the synchronization process; the API implementation should provide a session variable that stores the *since* value and that will be used in the add/update/remove methods.

When the second synchronization process (i.e. *SyncSession2*) starts, the *get_N_U_D* operation have a *where* clause: *since* > *t1*. The *get_N_U_D* method returns the items A, B, C and excludes the items 1 and 2 because they have the creation/update time equal to *t1*.

**Note**: if the *get_N_U_D* methods have a *where* clause: *since* > *t2*, the synchronization process has a point of failure because during *SyncSession1* the user could add an item (i.e. A) from an external User Interface.

### 7.2.2. endSync

Notify the JSON API that a sync has ended.

***Definition***

POST /<sync-source-name>/sync/end.

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

None.

***Response***

None.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error.

***Example***

Request:

POST /contact/sync/end.

## 7.3. Keys group

### 7.3.1. getAllSyncItemKeys

Retrieve the keys for all of the items in the SyncSource.

***Definition***

GET /<sync-source-name>/keys/all.

***Request Header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

None.

***Response***

An array with the item keys.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error.

***Example***

Request:

GET /contact/keys/all.

Response:

```
{"data":{
       "keys":[
               "key-1",
               "key-2",
               ...
               "key-n",
        ]
    }
}
```

## 7.3.2. getNewSyncItemKeys

Retrieve the keys for new items in the SyncSource since a specified time.

***Definition***

GET /<sync-source-name>/keys/new

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

- **since**: the time since the last sync in milliseconds on the sync server (Unix timestamp in milliseconds)
- **until**: the time of the current sync in milliseconds on the sync server (Unix timestamp in milliseconds)

***Response***

An array with the new item keys.

### *Errors*

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_INVALID_TIMESTAMP | The given sync type is not supported; the parameters *since* and *until* specify the invalid timestamp. | {"error":{ "code":"ERR_INVALID_TIMESTAMP", "message":"Invalid timestamp", "parameters":[ { "since":"12280808102500" }, { "until":"xxxxx" }] } } |

### *Example*

Request:

GET /contact/keys/new?since=123456789&until=133456789

Response:

```
{"data":{
        "keys":[
                "key-1",
                "key-2",
                ...
                "key-n",
        ]
    }
}
```

## 7.3.3. getUpdatedSyncItemKeys

Retrieve the keys for updated items in the SyncSource since a specified time.

### *Definition*

GET /<sync-source-name>/keys/updated

### *Request Header*

- **authorization**: the session ID returned in a previous login call

### *Parameters*

- **since**: the time since the last sync in milliseconds on the sync server
- **until**: the time of the current sync in milliseconds on the sync server

### *Response*

An array with the updated item keys.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_INVALID_TIMESTAMP | The given sync type is not supported; the parameters *since* and *until* specify the invalid timestamp. | {"error":{ "code":"ERR_INVALID_TIMESTAMP", "message":"Invalid timestamp", "parameters":[ { "since":"20080808102500" }, { "until":"xxxxx" }] } } |

***Example***

Request:

GET /contact/keys/updated?since=123456789&until=133456789

Response:

```
{"data":{
       "keys":[
              "key-1",
              "key-2",
              ...
              "key-n",
       ]
    }
}
```

## 7.3.4. getDeletedSyncItemKeys

Retrieve the keys for deleted items in the SyncSource since a specified time.

***Definition***

GET /<sync-source-name>/keys/deleted

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

- **since**: the time since the last sync in milliseconds on the sync server (Unix timestamp in milliseconds)
- **until**: the time of the current sync in milliseconds on the sync server (Unix timestamp in milliseconds)

***Response***

An array with the updated item keys.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_INVALID_TIMESTAMP | The given sync type is not supported; the parameters *since* and *until* specify the invalid timestamp. | {"error":{ "code":"ERR_INVALID_TIMESTAMP", "message":"Invalid timestamp", "parameters":[ { "since":"20080808102500" }, { "until":"xxxxx" }] } } |

***Example***

Request:

GET /contact/keys/deleted?since=123456789&until=133456789

Response:

```
{"data":{
       "keys":[
              "key-1",
              "key-2",
              ...
              "key-n",
       ]
    }
}
```

## 7.3.5. getSyncItemKeysFromTwin

Retrieve the keys of items that are twins of the item passed in the body of the request.

***Definition***

POST /<sync-source-name>/keys/twins

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

None.

***Body***

The body of this request contains the item that must be checked to see if it has a duplicate. This allows the JSON backend to apply any specific logic needed to discover contacts that match the given item. However, it is strongly recommended to compare the following criteria and rules; diverting from such rules may break the compatibility with the devices.

Recommended Contact fields for the search criteria are:

- firstname
- lastname
- email address

Recommended Events fields for the search criteria are:

- subject
- startdate
- enddate
- location

Recommended Task the the fields for the search criteria are:

- subject
- startdate
- duedate

***Response***

An array with the twin item keys.

***Errors***

If an error occurs, a 406 HTTP error code is returned and the content of the response is a JSON error.

***Example***

Request:

POST /contact/keys/twins.

Request body:

```
{"data":{
        "item":{
        "firstName":"John",
                "lastName":"Doe",
                "emailaddress":"john@somewhere.com"
        }
    }
}
```

Response:

```
{"data":{
       "keys":[
               "key-1",
               "key-2",
               ...
               "key-n",
        ]
   }
}
```

**Note**: a similar API with the same name but which supported a inserting the twin search parameters in the request is now deprecated.

## 7.4. Items group

For all the item-If the backend returns a 500 HTTP error code, the behavior depends on the the configuration parameter StopSyncOnFatalError: if it is true, the sync is stopped, otherwise the item is rejected.

### 7.4.1. addItem

Add an item to the SyncSource.

***Definition***

POST /<sync-source-name>/items.

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

- **since**: the time since the last sync in milliseconds on the sync server

***Body***

Contains all the fields for the object.

**Note:** all fields are sent, so the API always has access to any available field for processing.

***Response***

The server key and state of the added item.

***Error***

If an error occurs while adding the item from the SyncSource, a 406 HTTP error code is returned.

If the backend returns a 500 HTTP error code, the behavior depends on the the configuration parameter StopSyncOnFatalError: if it is true, the sync is stopped, otherwise the item is rejected.

***Example***

Request:

POST /contact/items?since=123456789

Request body:

```
{"data":{
        "content-type":"application/json-card",
        "item": {
                "firstName: "John",
                "LastName": "Doe"
                ...
                "property-n": "value-n",
        }
   }
}
```

Response:

```
{"data":{
        "key": "new-server-key"
    }
}
```

### 7.4.2. updateItem

Update an existing item in the SyncSource.

#### *Definition*

PUT /<sync-source-name>/items/<resource-key>

#### *Request header*

- **authorization**: the session ID returned in a previous login call

#### *Parameters*

- **since**: the time since the last sync in milliseconds on the sync server

#### *Body*

Contains all the fields for the object

**Note**: all fields are sent, so the API always has access to any available field for processing.

#### *Response*

Empty.

#### *Error*

If an error occurs while updating the item from the SyncSource, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_NOT_FOUND | The  item was not found. | {"error":{ "code":"ERR_INT_FOUND", "message":"Item not found" } } |

#### *Example*

Request:

PUT /contact/items/1323?since=123456789

Request body:

```
{"data": {
        "content-type":"application/json-card",
        "item": {
                "firstName: "Stefano",
                "LastName": "Fornari"
                ...
                "property-n": "value-n",
        }
    }
}
```

Response:

Empty.

### 7.4.3. removeItem

Remove an item from the SyncSource.

#### *Definition*

DELETE /<sync-source-name>/items/<resource-key>

#### *Request header*

- **authorization**: the session ID returned in a previous login call

#### *Parameters*

- **since**: the time since the last sync in milliseconds on the sync server

#### *Response*

Empty.

#### *Error*

If an error occurs while removing the item from the SyncSource, a 406 HTTP error code is returned and the content of the response is a JSON error, unless the configuration parameter StopSyncOnFatalError is true. In the latter case, the sync is stopped. The following error is expected:

| *Error code* | *Description* | *Error object* |
|---|---|---|
| ERR_NOT_FOUND | The item was not found. | {"error": { "code":"ERR_INT_FOUND", "message":"Item not found" } } |

#### *Example*

Request:

DELETE /contact/items/1342?since=123456789

Response:

Empty.

### 7.4.4. removeAllItems

Remove all the items from the SyncSource. This method will be used in the "Refresh from client" procedure.

#### *Definition*

DELETE /<sync-source-name>/items

#### *Request header*

- **authorization**: the session ID returned in a previous login call

#### *Parameters*

- **since**: the time since the last sync in milliseconds on the sync server

#### *Response*

Empty.

#### *Error*

If an error occurs while removing all the items from the SyncSource, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| *Error code* | *Description* | *Error object* |
|---|---|---|
| ERR_NOT_FOUND | The given item was not found. | {"error":{ "code":"ERR_NOT_FOUND", "message":"Items not found", "parameters": [ { "keys":["key_1","key_2","key_3"] } ] } } |

#### *Example*

Request:

DELETE /contact/items?since=123456789

Response:

Empty.

### 7.4.5. getItem

Get the content of an item from SyncSource given the server key.

#### *Definition*

GET /<sync-source-name>/items/<resource-key>

#### *Request header*

- **authorization**: the session ID returned in a previous login call

***Parameters***

None.

***Response***

The content and state of the updated item.

***Error***

If an error occurs while getting the item from the SyncSource, a 406 HTTP error code is returned and the content of the response is a JSON error. The following error is expected:

| ***Error code*** | ***Description*** | ***Error object*** |
|---|---|---|
| ERR_NOT_FOUND | The given  item was not found. | {"error":{ "code":"ERR_INT_FOUND", "message":"Item not found" } } |

***Example***

Request:

GET /contact/items/1342

Response:

```
{"data": {
        "content-type":"application/json-card",
        "item": {
                "firstName: "Stefano",
                "LastName": "Fornari"
                ...
                "property-n": "value-n",
        }
   }
}
```

## 7.5. Utility group

### 7.5.1. getConfiguration

Get the following information:

- Local date and time of the server where the API and the back-end are installed.

  This information will be used in order to know if all the servers (Funambol Server, API server and back-end server) have exactly the same date and time.

- Timezone set by the user in the user interface (i.e. a web access client).

  This API is called in order to evaluate and properly handle recurring appointment items.

***Definition***

GET /config/time

***Request header***

- **authorization**: the session ID returned in a previous login call

***Parameters***

None.

***Response***

The time and the timezone.

***Error***

If an error occurs while getting the item from the SyncSource, a 406 HTTP error code is returned.

***Example***

Request:

GET /config/time

Response:

```
{"data": {
        "time": "20090114T140147",
        "tzid": "GMT"
   }
}
```

# 8. Api Testing Tool

## 8.1. Introduction

This guide is intended for developers working on implementing the Funambol json API.

Using Funambol json testing tool the developer will be able to test the insert/remove/update and synchronization features of the server side developed api.

## 8.2. Installing

Funambol JSON testing tool is distributed as a .zip package, in order to use it you must unzip it, the scripts to run the application are placed under Funambol/json-test-api/bin/ . Executing the startup script will immediately start the tests.

#### *Linux*

In linux systems  the zip package can be unzipped using the “unzip” tool: *unzip Funambol-JSON-test-api-1.0.0.zip* . In order to start the tests the user should go th bin directory (cd Funambol/json-test-api/bin/ ) and run the start script ( *./run.sh* )

#### *Windows*

In windows it's possible to use any compression tool that supports “zip” files,  when extracted the files the user should locate the “Funambol” directory and then reach the startup scripts directory ( *cd Funambol\json-test-api\bin* ), in order to run the tests the user should execute the start script ( *run.cmd* )

## 8.3. Configuring

Under the “config” dir you will find two files: one for logging configurations (log4j.xml) and the other for server related properties.

### 8.3.1. Logging

In the logging file it's possible to configure the log that will appear in the console and will be recorded in the log file. The configuration follows the log4j standard : http://logging.apache.org/log4j/1.2/

#### *raising verbosity*

By default the log level is set to info, in this mode the user will only see the tests being executed and the errors(if any).

```
<category name="funambol.json-test-api">
       <priority value="info"/>
       <appender-ref ref="json-test-api-file"/>
   </category>
   <category name="funambol.json-test-api">
```

```
            <priority value="info"/>
            <appender-ref ref="json-test-api-console"/>
        </category>
```

While developing the server side json API it's useful to set the log level to “trace”

```
        <category name="funambol.json-test-api">
            <priority value="trace"/>
            <appender-ref ref="json-test-api-file"/>
        </category>
        <category name="funambol.json-test-api">
            <priority value="trace"/>
            <appender-ref ref="json-test-api-console"/>
        </category>
```

This way you will be able to see all the data exchanged with the server(sessionid's, request bodys request headers etc)

### 8.3.2. Server properties

Here you will be able so set up your server settings: server url, username and password

```
server-uri=http://localhost:8080/jsonsync
username=your_username
password=your_password
method=standard
```

#### *Method property*

The method property in the server.properties file is used in order to support two different server implementations. This happens because some backends might not be able to differentiate between new and updated items, if this is the case both new and updated items will be returned as new items and therefore method property should be set to “alternate” otherwise “standard” should be used.

## 8.4. Tests

The Funambol JSON tool ships with 15 default tests per source, in order to run this tests you just have to configure you server definitions and run the init script.

NOTE: Running all the tests (60 considering 15 per source) will take 25-30 minutes. See bellow how to turn off tests, if you want to run just a sub-set of the tests.

### 8.4.1. Structure

All the tests are placed under “testcases” directory, and all directories inside “testcases” will be consider as a source in server side (eg. testcases/contacts , testcases/notes, etc). Any directory inside a source will be considered the test name. Inside each directory representing a test name are the test files.

```
testcases
    contacts [this is the syncsource name server side]
        integrity1
            operations.properties
            item1.json
            item2.json
        sync1
            operations.properties
            item1.json
            item1_added.json
    events
        0001_test
            operations.properties
            item1.json
            item2.json
```

the “item*.json” files contain a json item that should have the item type of the source beeing tested, the “item*_added”, “item*_updated” and “item*_removed” contain a item key that can be used as a reference for some operations(more info bellow).

### 8.4.2. Turning off tests

If you don't have a complete server side implementation of the JSON API you will need to turn off some tests, you can do this by changing the source our test name from “sourcename” to “_sourcename” in this case the source/test wont be executed.

### 8.4.3. Writing new tests

There are the 2 ways to reference items: <instruction>:item1 and <instruction>:item1_added/<instruction>:item1_updated the first one refers to a json representation of an item, the second refers to a file containing a key(with is created when an item is inserted or updated)

| Instruction | Description | Syntax |
|---|---|---|
| login | performs the login in the json server | login |
| logout | logs out from the json server | logout |
| beginSync | tell the server that a sync session is going to begin | begynSync:<syncType> |

| | | |
|---|---|---|
| | | allowed types : “two-way”, “full”, “one-way-from-server”, “one-way-from-client”, “refresh-from-server”, “refresh-from-client” |
| endSync | tell the server that a sync session had ended. | endSync |
| getAllSyncItemKeys | get all the keys for the items in the server. | getAllSyncItemKeys |
| getDeletedSyncItemKeys | get all the keys of the deleted items since a specified time. | getDeletedSyncItemKeys |
| getUpdatedSyncItemKeys | get all the keys of the updated items since a specified time. | getUpdatedSyncItemKeys |
| getNewSyncItemKeys | get all the keys of the new items since a specified time. | getNewSyncItemKeys |
| getSyncItemFromId | get the item associated with a specific key. | GetSyncItemFromId:<key_file> |
| removeSyncItem | remove the item with a specific key. | removeSyncItem:<key_file> |
| addSyncItem | insert a specified item in the server and return it with a key. | addSyncItem:<item_file> |
| updateSyncItem | update a specified item in the server for a specific key. | updateSyncItem:<key_file>:<item_file> |
| getSyncItemKeysFromTwin | search for application defined equivalent items for a specified item. The key of the second parameter must be one of the returned twin items | getSyncItemKeysFromTwin:<item_fike:<key_file> |
| compareClientServer | compares the item on client with the item on the server the item returned by the server must have all the same keys and values as the item on client side | compareClientServer:<key_file>:<item_file> |
| deleteAllItems | Deletes all the items server side, for the source being tested | deleteAlltems |

All the commands described could be inserted in the “operation.properties” file that must be inside any test directory, also all the resources needed during the test (.json files) should be present in this directory.

## 8.5. Running the tests

As stated before In order to start the tests, the startup scripts under bin/ need to be executed. According to the log level the user will see only the test names and error or all the communications.

NOTE: in order to have correct synchronization results the testing framework MUST run on the same machine as the one where the testing API is running.

# 9. Appendix A – Contact examples

## 9.1. Contact

Below is an example of a JSON script that the Funambol Server sends to the back-end system. Note that there is no *key* field; when the back-end system returns a JSON item to the Funambol Server, the item must contains the *key* field.

```
{ "data" : {
        "content-type" : "application/json-card",
        "item" : {

                "firstName": "John",
                "lastName": "Smith"

                "businessAddressStreet":"via industria 12",
                "businessAddressCity":"trivolzio",
                "businessAddressState":"pavia",
                "businessAddressCountry":"italy",
                "businessAddressPostalCode":"27010",

                "homeAddressStreet":"23 rue marine",
                "homeAddressCity":"paris",
                "homeAddressState":"paris",
                "homeAddressCountry":"france",
                "homeAddressPostalCode":"10101",

                "email": "john.smith@email.ti",
                "email2": "allora@google.com",
                "email3": "thirdemail@email.ti",

                "phoneBusiness":"1111111",
                "phoneBusinessfax":"555555",
                "phoneHome":"222222",
                "phoneMobile":"3498888888",
                "phoneMobileHome" : "35512312312”,
                "phoneMobileBusiness" : "3556767678",

                "jobTitle":"engineer",
                "assistantName":"assistente",
                "nickName":"gioci",
                "spouseName":"mogliettina"
                "anniversary":"2000-06-27",
                "birthday":"1971-03-27",
                "url":"url.tre.com",
                "instantMessenger1":"ronny12",
                "company": "maruzzella srl",
                "department":"departement",
                "office":"12b"
        }
}}
```

# 10. Appendix B – Appointment examples

## 10.1. Single appointment

Below is an example of a JSON data structure containing information for a single appointment (type: single):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "holiday",
        "tzid": "America/Mexico_City",
        "startDate": "20081001T000000Z",
        "endDate": "20081001T000000Z",
        "location": "Acapulco",
        "allDay": false,
        "isRecurring": false
 }
}}
```

## 10.2. All day appointment

Below is an example of a JSON data structure containing information for a single all-day appointment (type: single, all day):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "holiday all day",
        "tzid": "America/Mexico_City",
        "allDay": true,
        "startDate": "20081001",
        "endDate": "20081001",
        "location": "Acapulco",
        "isRecurring": false
 }
}}
```

## 10.3. Daily recurring appointment

Below is an example of a JSON data structure containing information for a daily recurring appointment (type: daily; every 2 days; from 2008-10-01 to 2008-10-31):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20081001T170000Z",
        "endDate": "20081001T180000Z",
        "location": "Acapulco",
        "reminder": 1,
        "reminderTime": "15",                  -> mins before the appointment

        "isRecurring": true,
        "recurrenceType": "0",                 -> see the Constants paragraph.
        "interval": "2",                       -> in the Web UI "every 2 days"
        "patternStartDate": "20081001T110000", -> note the LocalTime
        "noEndDate": false,                    -> there's pattern end date
        "patternEndDate": "20081031T120000",   -> note the LocalTime

        "dayOfWeekMask": "0",            -> optional; not considered in this recurring
        "monthOfYear": "0",              -> optional; not considered in this recurring
        "dayOfMonth": "0",               -> optional; not considered in this recurring
        "instance": "0"                  -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.4. Weekly recurring appointment

Below is an example of a JSON data structure containing information for a weekly recurring appointment (type: weekly; every 1 week; every Wednesday; from 2008-07-20 to 2008-08-31)

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20080720T170000Z",
        "endDate": "20080720T180000Z",
        "location": "Acapulco",
        "reminder": 1,
        "reminderTime": "15",                  -> mins before the appointment

        "isRecurring": true,
        "recurrenceType": "1",                 -> see the Constants paragraph.
        "interval": "1",                       -> in the Web UI "every 1 week"
        "dayOfWeekMask": "8",                  -> see the Constants paragraph.
        "patternStartDate": "20080720T110000", -> note the LocalTime
        "noEndDate": false,                    -> there's pattern end date
        "patternEndDate": "20080831T120000",   -> note the LocalTime

        "monthOfYear": "0",              -> optional; not considered in this recurring
        "dayOfMonth": "0",                -> optional; not considered in this recurring
        "instance": "0"                   -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.5. Weekly recurring appointment

Below is an example of a JSON data structure containing information for a weekly recurring appointment (type: weekly; every 1 week; every Monday, Wednesday and Friday; from 2008-07-20 to 2008-08-31):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20080720T170000Z",
        "endDate": "20080720T180000Z",
        "location": "Acapulco",

        "isRecurring": true,
        "recurrenceType": "1",                  -> see the Constants paragraph.
        "interval": "1",                        -> in the Web UI "every 1 week"
        "dayOfWeekMask": "42",                  -> see the Constants paragraph.
        "patternStartDate": "20080720T110000",  -> note the LocalTime
        "noEndDate": false,                     -> there's pattern end date
        "patternEndDate": "20080831T120000",    -> note the LocalTime

        "monthOfYear": "0",               -> optional; not considered in this recurring
        "dayOfMonth": "0",                -> optional; not considered in this recurring
        "instance": "0"                   -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.6. All day with end weekly recurring appointment (converted from SIF)

Below is an example of a JSON data structure containing information for an all-day weekly recurring appointment with end date (type: weekly, all day; every 1 week; every Monday, Wednesday, Friday; from 2009-01-26 to 2009-03-10):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20090126",
        "endDate": "20090126",
        "location": "Acapulco",
        "allDay": true,

        "isRecurring": true,
        "recurrenceType": "1",                 -> see the Constants paragraph.
        "interval": "1",                       -> in the Web UI "every 1 week"
        "dayOfWeekMask": "42",                 -> see the Constants paragraph.
        "patternStartDate": "20090126T000000", -> note the LocalTime
        "noEndDate": false,                    -> there's pattern end date
        "occurrences": 20,
        "patternEndDate": "20090310T000000",   -> note the LocalTime

        "monthOfYear": "0",              -> optional; not considered in this recurring
        "dayOfMonth": "0",               -> optional; not considered in this recurring
        "instance": "0"                  -> optional; not considered in this recurring
  }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.7. All day without end weekly recurring appointment (converted from SIF)

Below is an example of a JSON data structure containing information for an all-day weekly recurring appointment without end date (type: weekly, all day; every 1 week; every Monday, Wednesday, Friday; from 2009-01-26):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "Kick off meeting",
       "tzid": "America/Mexico_City",
       "startDate": "20090126",
       "endDate": "20090126",
       "location": "Acapulco",
       "allDay": true,

       "isRecurring": true,
       "recurrenceType": "1",                 -> see the Constants paragraph.
       "interval": "1",                       -> in the Web UI "every 1 week"
       "dayOfWeekMask": "42",                 -> see the Constants paragraph.
       "patternStartDate": "20090126T000000",
       "noEndDate": true,                     -> there's pattern end date
       "occurrences": -1,

       "monthOfYear": "0",              -> optional; not considered in this recurring
       "dayOfMonth": "0",                -> optional; not considered in this recurring
       "instance": "0"                   -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.8. All day with end weekly recurring appointment (converted from vCalendar / iCalendar)

Below is an example of a JSON data structure containing information for an all-day weekly recurring appointment with end date (type: weekly, all day; every 1 week; every Monday, Wednesday, Friday; from 2009-01-26 to 2009-03-10):

**Note**: the user can create this kind of item with a Symbian (e.g. Nokia) device selecting an appointment "meeting" (X-EPOCAGENDAENTRYTYPE : APPOINTMENT) with start date: 00:00 and end date: 23:59.

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "Kick off meeting",
       "tzid": "America/Mexico_City",
       "startDate": "20090126",
       "endDate": "20090126",
       "location": "Acapulco",
       "allDay": true,

       "isRecurring": true,
       "recurrenceType": "1",                  -> see the Constants paragraph.
       "interval": "1",                        -> in the Web UI "every 1 week"
       "dayOfWeekMask": "42",                  -> see the Constants paragraph.
       "patternStartDate": "20090126T000000"   -> JSON Connector fixes the
                                                  format: 2009-01-26
       "noEndDate": false,                     -> there's pattern end date
       "occurrences": 20,
       "patternEndDate":"20090310T235900"      -> JSON Connector fixes the
                                                  format: 20090310T235900Z

       "monthOfYear": "0",               -> optional; not considered in this recurring
       "dayOfMonth": "0",                -> optional; not considered in this recurring
       "instance": "0"                   -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.9. All day without end yearly recurring appointment (converted from vCalendar / iCalendar)

Below is an example of a JSON data structure containing information for an all-day yearly recurring appointment without end date (type: all day, yearly; from 2009-12-25):

**Note**: the user can create this kind of item with a Symbian (e.g. Nokia) device  selecting an appointment "anniversary" (X-EPOCAGENDAENTRYTYPE : ANNIVERSARY).

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Happy Christmas",
        "tzid": "America/Mexico_City",
        "startDate": "20091225",
        "endDate": "20091225",
        "location": "Acapulco",
        "allDay": true,

        "isRecurring": true,
        "recurrenceType": "5",                 -> see the Constants paragraph.
        "interval": "1",                       -> in the Web UI "every 1 week"
        "monthOfYear": "12",
        "dayOfMonth": "25",
        "patternStartDate":"20091225T000000"    -> JSON Connector fixes the
                                                  format: 2009-12-25
        "noEndDate": true,                     -> there's pattern end date
        "occurrences": -1,
        "dayOfWeekMask": "0",            -> optional; not considered in this recurring
        "instance": "0"                  -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.10. Monthly recurring appointment (by day)

Below is an example of a JSON data structure containing information for a monthly recurring appointment (type: monthly; the 10th of  every 1 month; from 2008-10-01 to 2009-02-28):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20080720T170000Z",
        "endDate": "20080720T180000Z",
        "location": "Acapulco",

        "isRecurring": true,
        "recurrenceType": "2",                 -> see the Constants paragraph.
```

```
       "dayOfMonth": "10",                      -> in the Web UI "on 10 day"
       "interval": "1",                         -> in the Web UI "every 1 month"
       "patternStartDate": "20080720T110000",   -> note the LocalTime
       "noEndDate": false,                      -> there's pattern end date
       "patternEndDate": "20080831T120000",     -> note the LocalTime

       "monthOfYear": "0",                -> optional; not considered in this recurring
       "dayOfWeekMask": "0",              -> optional; not considered in this recurring
       "instance": "0"                    -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.11. Monthly recurring appointment (by position)

Below is an example of a JSON data structure containing information for a monthly recurring appointment (type: monthly; the first Thursday of every 1 month; from 2008-10-01 to 2009-02-28):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "Kick off meeting",
       "tzid": "America/Mexico_City",
       "startDate": "20080720T170000Z",
       "endDate": "20080720T180000Z",
       "location": "Acapulco",

       "isRecurring": true,
       "recurrenceType": "3",                   -> see the Constants paragraph.
       "interval": "1",                         -> in the Web UI "every 1 month"
       "instance": "1",                         -> first
       "dayOfWeekMask": "16",                   -> see the Constants paragraph.
       "patternStartDate": "20080720T110000",   -> note the LocalTime
       "noEndDate": false                       -> there's pattern end date
       "patternEndDate": "20080831T120000",     -> note the LocalTime

       "monthOfYear": "0",                -> optional; not considered in this recurring
       "dayOfMonth": "0"                  -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.12. Monthly recurring appointment (by position)

Below is an example of a JSON data structure containing information for a monthly recurring appointment (type: monthly; the last Friday of every 1 month; from 2008-10-01 to 2009-02-28):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20080720T170000Z",
        "endDate": "20080720T180000Z",
        "location": "Acapulco",

        "isRecurring": true,
        "recurrenceType": "3",                 -> see the Constants paragraph.
        "interval": "1",                       -> in the Web UI "every 1 month"
        "instance": "5",                       -> last
        "dayOfWeekMask": "32",                 -> see the Constants paragraph.
        "patternStartDate": "20080720T110000",  -> note the LocalTime
        "noEndDate": false,                    -> there's pattern end date
        "patternEndDate": "20080831T120000",    -> note the LocalTime

        "monthOfYear": "0",              -> optional; not considered in this recurring
        "dayOfMonth": "0"                -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.13. Monthly recurring appointment (by position)

Below is an example of a JSON data structure containing information for a monthly recurring appointment (type: monthly; the second weekend day (Saturday, Sunday) every 1 month; from 2008-10-01 to 2009-02-28):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "Kick off meeting",
        "tzid": "America/Mexico_City",
        "startDate": "20080720T170000Z",
        "endDate": "20080720T180000Z",
        "location": "Acapulco",

        "isRecurring": true,
        "recurrenceType": "3",                 -> see the Constants paragraph.
        "interval": "1",                       -> in the Web UI "every 1 month"
        "instance": "2",                       -> second
        "dayOfWeekMask": "65",                 -> see the Constants paragraph.
        "patternStartDate": "20080720T110000",  -> note the LocalTime
        "noEndDate": false,                    -> there's pattern end date
        "patternEndDate": "20080831T120000",    -> note the LocalTime
```

```
       "monthOfYear": "0",               -> optional; not considered in this recurring
       "dayOfMonth": "0"                 -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.14. Monthly recurring appointment (by position)

Below is an example of a JSON data structure containing information for a monthly recurring appointment (type: monthly; the second weekday (Monday, Tuesday, Wednesday, Thursday, Friday) every 1 month; from 2008-10-01 to 2009-02-28):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "Kick off meeting",
       "tzid": "America/Mexico_City",
       "startDate": "20080720T170000Z",
       "endDate": "20080720T180000Z",
       "location": "Acapulco",

       "isRecurring": true,
       "recurrenceType": "3",                  -> see the Constants paragraph.
       "interval": "1",                        -> in the Web UI "every 1 month"
       "instance": "2",                        -> second
       "dayOfWeekMask": "62",                  -> see the Constants paragraph.
       "patternStartDate": "20080720T110000",  -> note the LocalTime
       "noEndDate": false,                     -> there's pattern end date
       "patternEndDate": "20080831T120000",    -> note the LocalTime

       "monthOfYear": "0",               -> optional; not considered in this recurring
       "dayOfMonth": "0"                 -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.15. Yearly recurring appointment (by day)

Below is an example of a JSON data structure containing information for a yearly recurring appointment (type: yearly; the 7th of October every 1 year; from 2008-10-01):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
       "subject": "birthday",
       "tzid": "America/Mexico_City",
       "startDate": "20080720T170000Z",
       "endDate": "20080720T180000Z",
       "location": "Acapulco"

       "isRecurring": true,
       "recurrenceType": "5",                -> see the Constants paragraph.
       "interval": "1",                      -> in the Web UI "every 1 month"
       "dayOfMonth": "7",                    -> oct. 7th
       "monthOfYear": "10",                  -> oct.
       "patternStartDate": "20080720T110000",  -> note the LocalTime
       "noEndDate": true,                    -> there's no pattern end date

       "instance": "0",                -> optional; not considered in this recurring
       "dayOfWeekMask": "0"            -> optional; not considered in this recurring
}
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

## 10.16. Yearly recurring appointment (by position)

Below is an example of a JSON data structure containing information for a yearly recurring appointment (type: yearly; the second Monday in October; from 2008-10-01):

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "birthday",
        "tzid": "America/Mexico_City",
        "startDate": "20081001T170000Z",
        "endDate": "20081001T180000Z",
        "location": "Acapulco",

        "isRecurring": true,
        "recurrenceType": "6",                -> see the Constants paragraph.
        "interval": "1",                      -> in the Web UI "every 1 month"
        "dayOfWeekMask": "2",                 -> see the Constants paragraph.
        "instance": "2",                      -> see the Constants paragraph.
        "monthOfYear": "10",                  -> oct.
        "patternStartDate": "20081001T110000",  -> note the LocalTime
        "noEndDate": true,                    -> there's no pattern end date

        "dayOfMonth": "0"               -> optional; not considered in this recurring
 }
}}
```

**Note**: the *occurrences* value could be evaluated starting from *patternStartDate* and *patternEndDate.* The API could send both *occurrences* and *patternEndDate*, but they have the same meaning.

# 11. Appendix C – Task examples

## 11.1. Single task

Below is an example of a JSON data structure containing information for a single task:

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "birthday",
        "tzid": "America/Mexico_City",
        "allDay" : true,
        "startDate": "20081001",
        "dueDate": "20081001",

        "complete": false,
        "status": "0",
        "percentComplete": "0",
        "remider": false
 }
}}
```

## 11.2. Task with reminder

Below is an example of a JSON data structure containing information for a task with reminder:

```
{ "data" : {
  "content-type" : "application/json-appointment",
  "item" : {
        "subject": "birthday",
        "tzid": "America/Mexico_City",
        "allDay" : true,
        "startDate": "20081001",
        "dueDate": "20081001",

        "reminder": true,
        "reminderDate": "20090129T100000",

        "complete": false,
        "status": "0",
        "percentComplete": "0",
        "remider": false
 }
}}
```

# 12. Appendix D – Constants

## 12.1. Constants used in the recurring appointment format

#### *DaysOfWeekMask*

```
Sunday        = 1;
Monday        = 2;
Tuesday       = 4;
Wednesday     = 8;
Thursday      = 16;
Friday         = 32;
Saturday      = 64;
```

#### *RecurrenceType*

```
RecursDaily    = 0;
RecursWeekly   = 1;
RecursMonthly  = 2;
RecursMonthNth = 3;
RecursYearly   = 5;
RecursYearNth  = 6;
```

#### *Instance*

```
first   = 1;
second  = 2;
third   = 3;
fourth  = 4;
last    = 5;
```

## 12.2. Constants used in the appointment format

#### *Sensitivity*

```
PUBLIC
PRIVATE
CONFIDENTIAL
X-PERSONAL
```

**Note**: if the value of the *sensitivity* property is different than one of the strings specified above, it will be mapped to *X-PERSONAL*.

#### *BusyStatus*

```
Free          = 0;
Tentative     = 1;
Busy          = 2;
OutOfOffice   = 3;
```

#### *Importance*

```
Low           = 0;
Normal        = 1;
High          = 2;
```

## 12.3. Constants used in the task format

### *Task completion status*

```
accepted     = 0
sent         = 1
tentative    = 2
in-process   = 3
confirmed    = 4
completed    = 5
needs-action = 6
declined     = 8
```

# 13. Appendix E – Contact mapping vCard / extended JSON

The following table shows the mapping of the fields between the vCard and the extended format of the JSON Contact object.

| ***JSON Extended Format Fields*** | ***VCARD Field*** |
|---|---|
| "folder" | X-FUNAMBOL-FOLDER |
| "title" | N:LastName;FirstName;MiddleName;**Salutation**;Suffix |
| "firstName" | N:LastName;**FirstName**;MiddleName;Salutation;Suffix |
| "middleName" | N:LastName;FirstName;**MiddleName**;Salutation;Suffix |
| "lastName" | N:**LastName**;FirstName;MiddleName;Salutation;Suffix |
| "suffix" | N:LastName;FirstName;MiddleName;Salutation;**Suffix** |
| "email" | EMAIL;INTERNET: |
| "email2" | EMAIL;INTERNET;HOME |
| "email3" | EMAIL;INTERNET;WORK |
| "imAddress" | EMAIL;INTERNET;HOME;X-FUNAMBOL-INSTANTMESSENGER |
| "businessAddressStreet" | ADR;WORK:PO box;Extended Address;**Address**;City;State;PostalCode;Country |
| "businessAddressCity" | ADR;WORK:PO box;Extended Address;Address;**City**;State;PostalCode;Country |
| "businessAddressCountry" | ADR;WORK:PO box;Extended Address;Address;City;State;PostalCode;**Country** |
| "businessAddressState" | ADR;WORK:PO box;Extended Address;Address;City;**State**;PostalCode;Country |
| "businessAddressPostalCode" | ADR;WORK:PO box;Extended Address;Address;City;State;**PostalCode**;Country |
| "businessAddressPostOfficeBox" | ADR;WORK:**PO box**;Extended Address;Address;City;State;PostalCode;Country |
| "businessAddressExtendedAddress" | ADR;WORK:PO box;**Extended Address**;Address;City;State;PostalCode;Country |
| "homeAddressStreet" | ADR;HOME:PO box;Extended Address;**Address**;City;State;PostalCode;Country |
| "homeAddressCity" | ADR;HOME:PO box;Extended Address;Address;**City**;State;PostalCode;Country |
| "homeAddressState" | ADR;HOME:PO box;Extended Address;Address;City;**State**;PostalCode;Country |
| "homeAddressCountry" | ADR;HOME:PO box;Extended Address;Address;City;State;PostalCode;**Country** |
| "homeAddressPostalCode" | ADR;HOME:PO box;Extended Address;Address;City;State;**PostalCode**;Country |
| "homeAddressPostOfficeBox" | ADR;HOME:**PO box**;Extended Address;Address;City;State;PostalCode;Country |
| "homeAddressExtendedAddress" | ADR;HOME:PO box;**Extended Address**;Address;City;State;PostalCode;Country |
| "otherAddressStreet" | ADR:PO box;Extended Address;**Address**;City;State;PostalCode;Country |
| "otherAddressCity" | ADR:PO box;Extended Address;Address;**City**;State;PostalCode;Country |
| "otherAddressState" | ADR:PO box;Extended Address;Address;City;**State**;PostalCode;Country |
| "otherAddressCountry" | ADR:PO box;Extended Address;Address;City;State;PostalCode;**Country** |
| "otherAddressPostalCode" | ADR:PO box;Extended Address;Address;City;State;**PostalCode**;Country |
| "otherAddressPostOfficeBox" | ADR:**PO box**;Extended Address;Address;City;State;PostalCode;Country |
| "otherAddressExtendedAddress" | ADR:PO box;**Extended Address**;Address;City;State;PostalCode;Country |
| "phoneAssistant" | AGENT:TEL;WORK |
| "phoneBusiness" | TEL;VOICE;WORK: |
| "phoneBusiness2" | TEL;VOICE;WORK: |
| "phoneBusinessFAX" | TEL;WORK;FAX: |
| "phoneCallback" | TEL;X-FUNAMBOL-CALLBACK: |
| "phoneCar" | TEL;CAR;VOICE: |
| "phoneCompany" | TEL;WORK;PREF |
| "phoneHome" | TEL;VOICE;HOME: |
| "phoneHome2" | TEL;VOICE;HOME: |
| "phoneHomeFAX" | TEL;HOME;FAX: |
| "phoneMobile" | TEL;CELL: |
| "phoneMobileHome" | TEL;CELL;HOME: |
| "phoneMobileBusiness" | TEL;CELL;WORK: |
| "phoneOther" | TEL;VOICE: |
| "phoneOtherFAX" | TEL;FAX: |
| "phonePrimary" | TEL;PREF;VOICE |
| "phonePager" | TEL;PAGER |
| "phoneRadio" | TEL;X-FUNAMBOL-RADIO |

| | |
|---|---|
| "phoneTelex" | TEL;X-FUNAMBOL-TELEX |
| "url" | URL |
| "instantMessenger1" | Not supported yet |
| "body" | NOTE |
| "jobTitle" | TITLE |
| "company" | ORG:**Company**;Department;Office |
| "sensitivity" | CLASS |
| "department" | ORG:Company;**Department**;Office |
| "office" | ORG:Company;Department;**Office** |
| "managerName" | X-FUNAMBOL-MANAGER |
| "assistantName" | AGENT:FN |
| "nickName" | NICKNAME |
| "spouseName" | X-FUNAMBOL-SPOUSE |
| "anniversary" | X-FUNAMBOL-ANNIVERSARY |
| "birthday" | BDAY |
| “displayName” | FN |
| "homeAddressLabel" | LABEL;HOME |
| "otherAddressLabel" | LABEL |
| "photo" | PHOTO |
| "photoType" | PHOTO:TYPE |
| "children" | CHILDREN |
| "gender" | X-FUNAMBOL-GENDER |
| "hobbies" | X-FUNAMBOL-HOBBIES |
| "urlHome" | URL:HOME |
| "businessAddressLabel" | LABEL;WORK |
| "profession" | ROLE |
| “urlWork” | URL:WORK |
| "language" | LANGUAGES |
| "importance" | PRIORITY |
| "subject" | X-FUNAMBOL-SUBJECT |
| "mileage" | X-FUNAMBOL-MILEAGE |

# 14. Appendix F – Event mapping vCal / extended JSON

The following table shows the mapping of the fields between the vCalendar format and the extended format of the JSON Appointment object.

| ***JSON Extended Format Fields*** | ***VCAL Field*** |
|---|---|
| "folder" | X-FUNAMBOL-FOLDER |
| "allDay" | X-FUNAMBOL-ALLDAY |
| "startDate" | DTSTART |
| "endDate" | DTEND |
| "tzid" | TZ, DAYLIGHT |
| "subject" | SUMMARY |
| "body" | DESCRIPTION |
| "location" | LOCATION |
| “organizer” | ORGANIZER |
| "reminder" | **AALARM**;TYPE=Type;VALUE=RunTime;SnoozeTime;RepeatCount;AudioContent: |
| "reminderTime" | AALARM;TYPE=Type;VALUE=**RunTime**;SnoozeTime;RepeatCount;AudioContent: |
| “reminderSoundFile” | AALARM;TYPE=Type;VALUE=RunTime;SnoozeTime;RepeatCount;**AudioContent**: |
| "busyStatus" | X-MICROSOFT-CDO-BUSYSTATUS |
| "categories" | CATEGORIES |
| "sensitivity" | CLASS |
| "importance" | PRIORITY |
| "dayOfMonth" | RRULE (*) |
| "dayOfWeekMask" | RRULE (*) |
| "instance" | RRULE (*) |
| "interval" | RRULE (*) |
| "isRecurring" | RRULE (*) |
| "monthOfYear" | RRULE (*) |
| "noEndDate" | RRULE (*) |
| "occurrences" | RRULE (*) |
| "exceptionsExcluded" | EXDATE |
| "exceptionsIncluded" | Not supported yet |
| "patternEndDate" | RRULE (*) |
| "patternStartDate" | RRULE (*) |
| "recurrenceType" | RRULE (*) |

(*) RRULE is a pattern specification for event recurrences, so the mapping with JSON object properties is not linear and depends on pattern values. See http://www.imc.org/pdi/ for vCalendar format specification.

# 15. References

[1] Funambol Developer’s Guide

[2] Funambol Installation and Administration Guide

[3] JSON Connector Design Document

[4] JSON, http://en.wikipedia.org/wiki/JSON

[5] REST, http://en.wikipedia.org/wiki/REST

[6] IETF - RFC 4627 (JSON)

[7] IETF - RFC 2426 (vCard)

[8] IETF - RFC 2445 (vCalendar 2.0 / iCalendar)