FUNAMBOL mobile open source

Windows Mobile Plug-in Design Document

May, 2007

Table of Contents

1. Change Control

Date	***Version***	***Author and Title***	***Change***
09.05.2007	0.1	Marco Magistrali	Initial draft
23.05.2007	0.2	Marco Magistrali	Added other parts
08.06.2007	0.3	Marco Magistrali	Modified configuration part (ClientSettings)

2. Introduction

This document designs how the Funambol Windows Mobile plug-in generally works and what it synchronizes through the syncML protocol: PIM data (contact, calendar, task), briefcase, notes and mail. It explains how the plug-in is built and what are its behaviors.

2.1. Audience

The following list shows the purposes this document can be used for:

- Information for all teams or groups of Funambol
- Overview for people interested in the project
- Overview for new and old developers to make them able to change software

2.2. Architecture overview

This paragraph shows graphically the interactions amongst the Windows Mobile components that can start a synchronization with the server [Figure 1]. The "S*ync runners*" layer is a group of actors that can start a sync in different ways: manually, using the buttons provided by the user interface, or automatically because of a scheduled sync or a server request message. The "S*ync logic*" layer contains all the functions needed to start the sync session, to exchange the data with the server and to modify the device databases according to the synchronization commands.
The full functionality of the "S*ync runners*" and the "S*ync logic*" layers will be described in the next sections.

The Pocket PC or Smartphone User Interface allow the user to start to configure all the needed parameters, as user name/password, server URL, which PIM database to synchronize and so on.
After these initializing procedure, the UI permits to sync all the selected database in a single shot or one at time.
There are several way to synchronize the device: the default used is the *two-way sync*, in which all the modified elements (new, updated and deleted) are exchanged to and from the server. The other synchronization types supported are:

- *one-way from client*: only the modifications from the client are exchanged in the sync process
- *one-way from server*: only the modifications from the server are exchanged in the sync process
- *refresh from client*: a clean copy of all the client items are stored on the server
- *refresh from server*: all the items on the client are deleted and a clean copy of all server items is stored

The latter two sync type are called recover sync, because they can fully recover all the data both on client or server side.

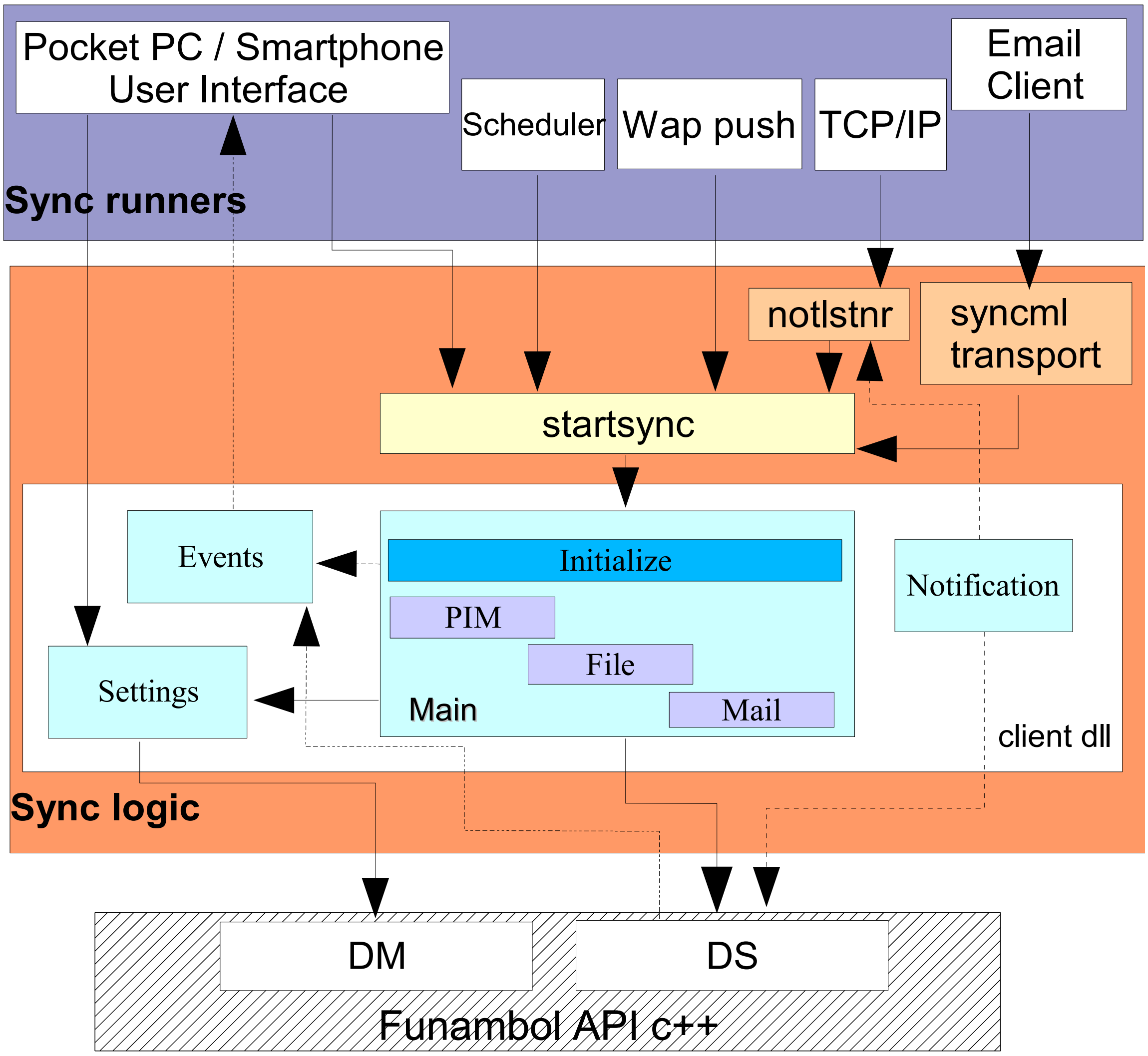


Figure 1: Funambol Windows Mobile plug-in general architecture

Through the setting parameters, the user can set also the scheduling time, so that a sync is started automatically at defined intervals, and the server notification mechanisms in which the SyncML server, using TCP/IP or wap push message (SMS), asks the client to start a sync.

For each method, scheduled sync or the server alerted sync, there is a dedicate client that starts the synchronization at need. In this case, no User Interface is shown and the sync works in background.

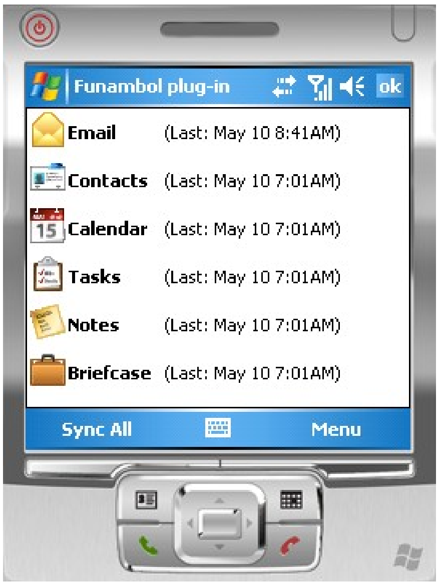
Additionally, the Funambol plug-in is hooked to the default email client with a a custom email account and a specific transport layer, similar to a POP3 or IMAP4 transport, that integrates the SyncML sync process.

3. Sync runners

In this section it will be analyzed the component that are responsible to start a synchronization process. There is the plug-in interface and the windows mobile mail interface that have a graphic way to start a sync. The user can press a button to force the sync to start. The scheduler and the notification via TCP/IP or SMS wap push are silent way to initialize the sync.

3.1. Pocket PC and Smartphone User Interface

The plug-in interface graphically is the same for both the pocket pc and smartphone version. The only difference is the smartphone version doesn't support the synchronization of Note data source. After the user installed the plug-in and runs, the UI looks like follow:



Email (Last: May 10 8:41AM)
Contacts (Last: May 10 7:01AM)
Calendar (Last: May 10 7:01AM)
Tasks (Last: May 10 7:01AM)
Notes (Last: May 10 7:01AM)
Briefcase (Last: May 10 7:01AM)

Figure 2: Funambol Windows Mobile Plug-in for Pocket PC User Interface

Basically the two menu buttons need:

- *SyncAll*: starts the sync of all enabled sources. In the case of Figure 2, all the sources are enabled. If one not enabled, the correspondent line is grayed.
- *Menu:* enables several forms to set the account parameters, the sources to sync, the notification listener, the log lever, the mail parameters and all other settings.

To learn more about how the plug-in works from user perspective see the Plug-in User guide [1].

The User Interface is developed with the support of **MFC** library (Microsoft Foundation Classes); windows and GUI objects extend basic MFC objects. The UI module mainly contains:

1. classes to implement forms and windows, for graphical interaction
2. methods to start/stop/end the sync process
3. methods linked to the message map, to execute actions capturing messages sent by dll library.

The general architecture of the User Interface is structured as follow:

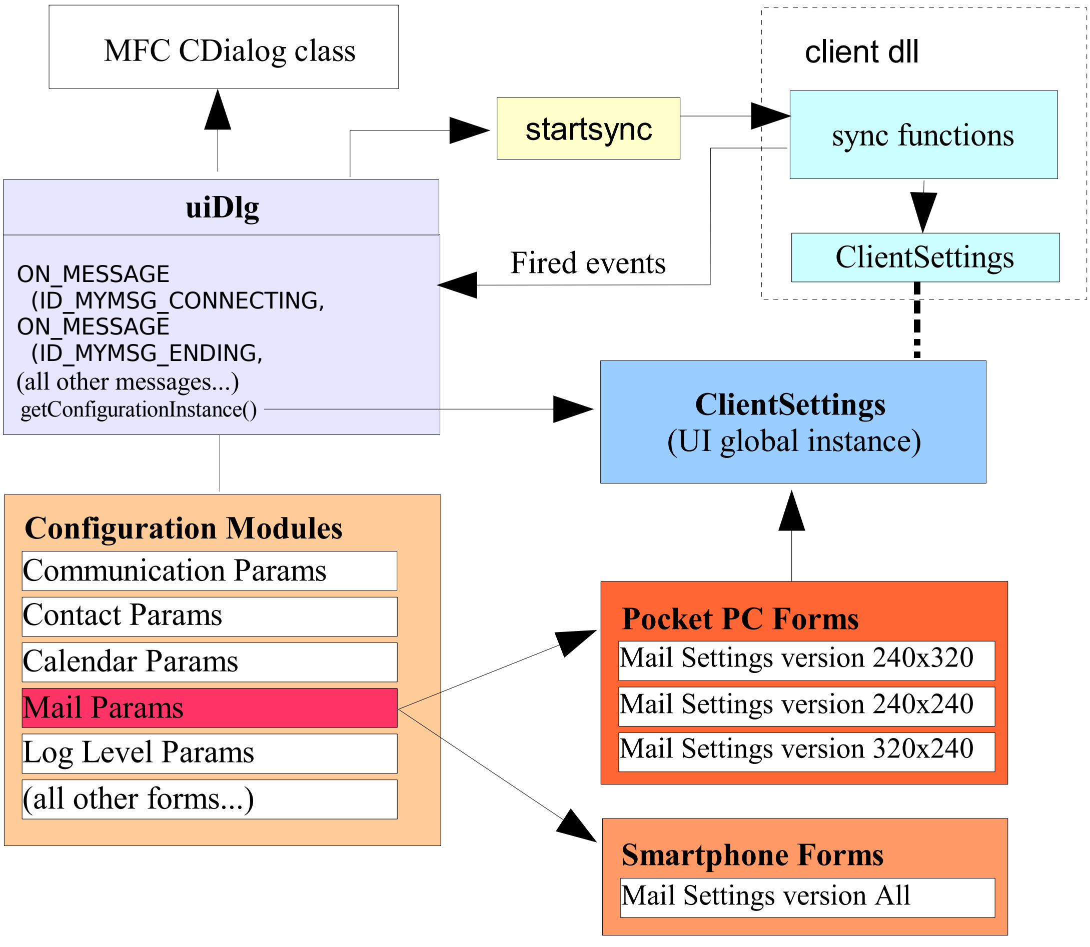


Figure 3: Funambol Windows Mobile plug-in general architecture

The *uiDlg* module is the class associated to the main form that is responsible to show to the user the list of the sources that can sync. When the user opens the plug-in, this form loads the configuration settings and choose which sources are to be enabled and which not. The configuration (*ClientSettings* module) is a singleton class that is created during the initialization process.
Every other forms uses the same instance of the configuration to populate their own fields: the “Save” menu button in these forms sets the modified value in the configuration instance and at the same time writes the value in the persistent registry tree. The configuration class is the same provided by the main dll library (see 4.1.5) and the UI handles its own instance.
Basically the source code associated to the graphical form is grouped in the classes showed in “Configuration Modules” in the Figure 3. Only some classes are represented in the figure. The differences between some specific code for pocket pc and for smartphone are handled using some macro that distinguishes on which device the application is targeting for.
An example is due to the behavior the pocket pc can sync note source too. So in the *uiDlg* form there is a piece of code:

```
#if defined(WIN32_PLATFORM_PSPC)

    // if the platform target is pocket pc: handling note source

#endif
```

Alternatively it is also possible to find

```
#if defined(WIN32_PLATFORM_WFSP)

    // if the platform target is smartphone: don't resize screen

#endif
```

The *resources* used for the different platforms, like icons or images, are shared in the same resource repository; however the forms containing the buttons, the labels, the text box and so on are duplicated for both platforms. More, for the pocket pc can exist more than one version for a same panel due to different screen size of the devices. The Figure 3 shows the Mail Settings that needs three different forms. If there are to much buttons, labels and checkbox is needed to repaint their position according to the device screen size. Currently, the supported dimensions are:

```
portrait:  w240 x h320
square:    w240 x h240
landscape: w320 x h240
```

For the smartphone device it was not necessary to add multiple form and the supported size are:

```
landscape: w320 x h240
small:     w176 x h220
QVGA:      w240 x h320
```

When the sync process is running, the main dll library fires some events about the current status of the sync process. These events (see 4.1.2) are sent as Windows Messages

```
SendMessage(wnd, ID_MYMSG_TOTAL_ITEMS,(WPARAM)-1, totalSize);
```

and they are captured by methods defined in the message map of *uiDlg* module. They basically are used to get the data to refresh the User Interface and give a feedback to the user.
The list of the mapping messages are listed below:

Name	***Description***

ID_MYMSG_CONNECTING (WM_APP+1)	OnMsgConnecting: the sync is started, the menu is refreshed, the icon animations start.
	WPARAM = NULL
	LPARAM = NULL

ID_MYMSG_ENDING (WM_APP+2)	OnMsgEndingSync: the sync is finished. The menu is refreshed, the icon animation stop
	WPARAM = NULL
	LPARAM = NULL

ID_MYMSG_STARTSYNC_ENDED (WM_APP+3)	OnStartsyncEnded: called by the startsync.exe program when the sync process ends, sending the last error code. The UI refresh the last sync timestamps. WPARAM = NULL LPARAM = error code of the sync process. 0 is success
ID_MYMSG_STARTING_SYNC (WM_APP+4)	OnStartingSync: called by the main dll, at the beginning of the sync, to signal the id of the first source to sync to understand where to show the first message (“Connecting...”) to the user in the UI. WPARAM = 1 LPARAM = the id of the first source to sync

ID_MYMSG_ENDING_SOURCE (WM_APP+5)	OnMsgEndingSource: called by the event of the main dll to signal the current syncing source is terminated.
	WPARAM = NULL
	LPARAM = NULL

ID_MYMSG_STARTING_SOURCE (WM_APP+6)	OnStartingSource: called by the event of the main dll to

	signal the current syncing source is starting. WPARAM = NULL LPARAM = id of the source to sync (retrieved by its source name)
ID_MYMSG_ITEM_SYNCED_FROM_SERVER (WM_APP+7)	OnMsgItemSyncedFromServer: called by the event of the main dll to signal an item is sent by the server. This item can be new, updated or deleted. For the mail source, this event is fired by the MailSyncSource. WPARAM = id of the source. 0 if the source is mail LPARAM = NULL
ID_MYMSG_TOTAL_ITEMS (WM_APP+8)	OnMsgTotalItems: It sets the total items that are exchanged in the current sync, both from client and server. It is the second value of 3/15 items. It is fired by the listener for total items from server. It is fired by the sync sources for the total items from client. WPARAM = id of the source. If -1 the items are from client LPARAM = number of total items
ID_MYMSG_ITEM_SYNCED_FROM_CLIENT WM_APP+9)	OnMsgItemSyncedFromClient: called by the event of the main dll to signal an item is sent by the client. This item can be new, updated or deleted. WPARAM = id of the source. If -0 if the source is mail LPARAM = NULL
ID_MYMSG_SOURCE_STATE (WM_APP+10)	OnMsgSourceState: called from the main dll at the end of the sync to notify if a source has was finished properly or not WPARAM = id of the source. LPARAM = (SYNCSOURCE_STATE_OK = 1 | SYNCSOURCE_STATE_NOT_SYNCED = 2)

At the end of the sync process, the configuration is refreshed to reload the parameters that are modified by the sync, like the last time the sync ended. Also the main UI is refreshed to reflect these changes.

An example of complete synchronization diagrams can be as follow:

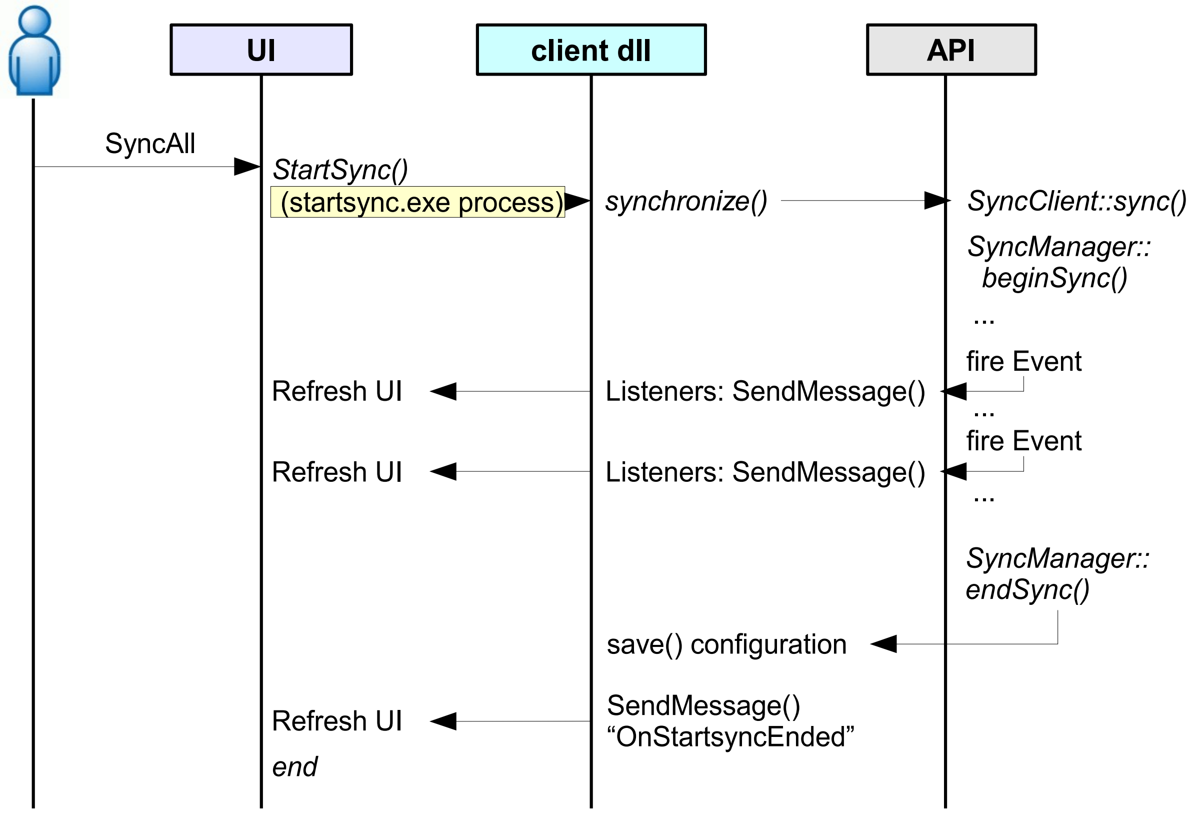


Figure 4: Sequence diagram of a complete synchronization

3.2. Email client

The email client is the usual program provided with the Windows Mobile to handle the user email accounts. The Funambol plug-in inserts a custom dll library to hook a new account type named “Funambol” (see 4.4). Through the plug-in installer a new mail account, also named “Funambol”, is created and used to synchronize the email data source with the server. This way to sync the email follows the user experience that already uses the device to read and write emails.

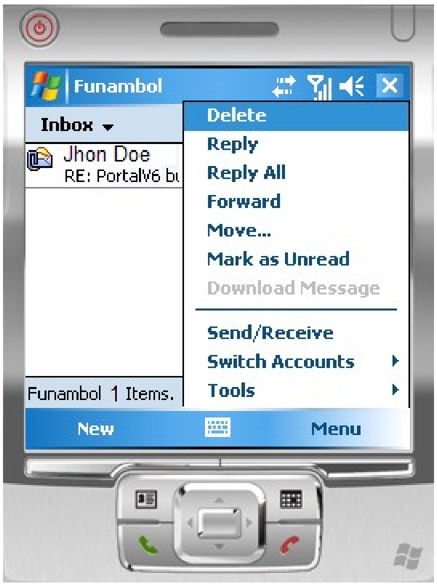


Figure 5: Windows Mobile embedded mail interface

The plug-in interacts with the email client through the custom library. So by clicking Send/Receive menu item the library is invoked and a new sync start, only for the email data source. It calls the

```
startsync.exe once mail
```

see the 4.2 paragraph about the startsync process.

3.3. Scheduler

The Funambol plug-in can be configured to perform a sync in background at a specified time, that can be set through the User interface configuration panel. The permitted scheduled intervals are

- Manual Sync
- After every 5 minutes
- After every 10 minutes
- After every 15 minutes
- After every 30 minutes
- After every hour
- After 2 hours
- After 4 hours
- After a day

To set a time after which the sync process starts, the plug-in uses a windows function ([2]) whose arguments are the name of the application to call and the time it will be called.

```
CeRunAppAtTime(TACHAR* appName, SYSTEMTIME &nextTime)
```

The appName is the “startsync.exe” program name (see 4.2) and the nextTime is the current time shifted of the minutes set by the user. At the `nextTime` the windows OS calls the `appName` with a particular argument `APP_RUN_AT_TIME`. Due to this call, the startsync.exe program does two main action: first of all it call the `CeRunAppAtTime` to set the next scheduled time and then it starts the sync process.
If the user wants to stop the schedule process, he sets “Manual Sync” from the list of choices: this means the `nextTime` is set to 0 so the schedule process won't be scheduled anymore.

3.4. Wap Push

The wap-push sync initializer is a method to start a sync process through a particular binary SMS message. It is possible to configure the plug-in to be able to wait and understand this particular SMS, and when it comes, the plug-in parses the message and it starts the sync process according with the information in the message.

The structure of the notification message, that is the same used also in the TCP/IP notification (see 3.5), is named package #0 and has the follow structure:

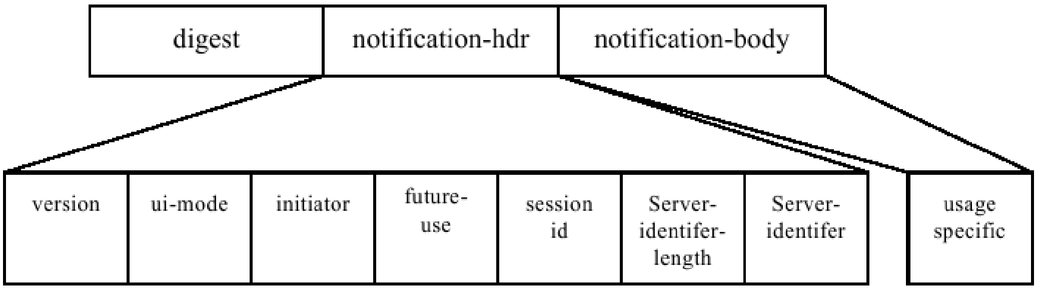


Figure 6: Format of the package #0 header

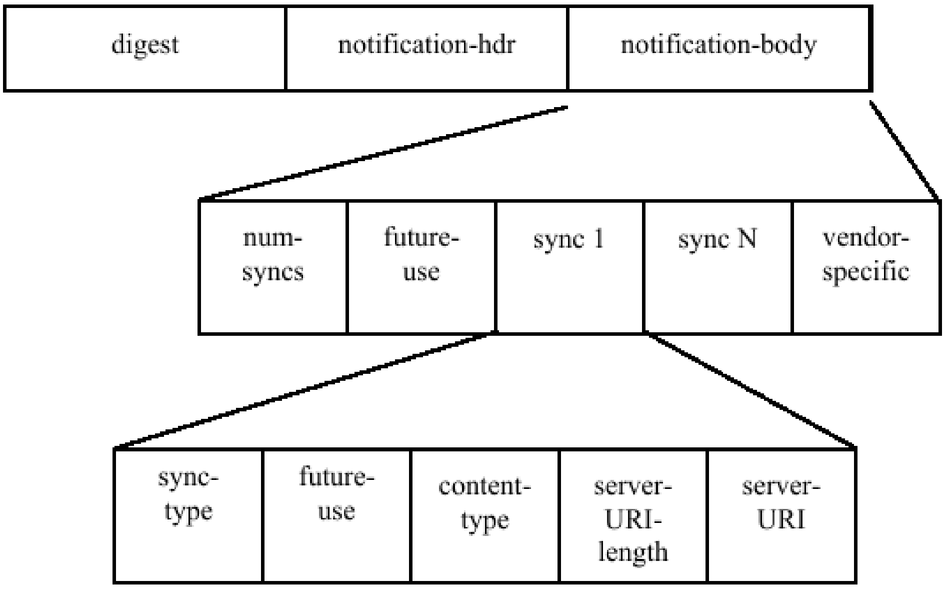


Figure 7: Format of the package #0 body

The following ABNF [RFC2234] defines the syntax for the package. The order and the size of the fields MUST be as specified in the following syntax of the Notification Package.

```
<digest> ::= 128*BIT                                            ; ‘MD5 Digest value’
<notification> ::= <notification-hdr><notification-body>
<notification-hdr> ::= <version><ui-mode><initiator><future-use>
                 <sessionid><server-identifer-length><server-identifier>
<version> ::= 10*BIT                                            ; ‘Protocol Version’
<ui-mode> ::= <not-specified> / <background> /                  ; ‘Background/Informative
                <informative> / <user-interaction>              ; User Interaction
                                                                ; session’
<not-specified> ::= “00”                                        ; ‘2*bit value “0”’
<background> ::= “01”                                           ; ‘2*bit value “1”’
<informative> ::= “10”                                          ; ‘2*bit value “2”
<user-interaction> ::= “11”                                     ; ‘2*bit value “3”’
<initiator> ::= <user> / <server>                               ; ‘Server/User initiated’
<user> ::= “0”                                                  ; ‘1*bit value “0”’
<server> ::= “1”                                                ; ‘1*bit value “1”’
<future-use> ::= 27*BIT                                         ; ‘Reserved for future
                                                                ; use’
<sessionid> ::= 16*BIT                                          ; ‘Session identifier’
<server-identifier-length> ::= 8*BIT                            ; ‘Server Identifier
                                                                ; length’
<server-identifier> ::= <server-identifier-length >*CHAR        ; ‘Server Identifier’
<notification-body> ::= <num-syncs><future-use>*<sync>
                          <vendor-specific>                     ; ‘Body Data’
<num-syncs> ::= 4*BIT                                           ; ‘Number of syncs’
<future-use> ::= 4*BIT                                          ; ‘Reserved for future
                                                                ; use’
<sync> ::= <sync-type><future-use><content-type>
            <server-URI-length><server-URI>                     ; ‘Sync Information’
<sync-type> ::= 4*BIT                                           ; ‘Synchronization type’
<future-use> ::= 4*BIT                                          ; ‘Reserved for future
                                                                ; use’
<content-type> ::= 24*BIT                                       ; ‘Content type’
<server-URI-length> ::= 8*BIT                                   ; ‘Server URI Length’
                                                                ; ‘Server URI
<server-URI> ::= n*BIT
<vendor-specific> ::= n*BIT                                     ; ‘Optional vendor-
                                                                ; specific information’
```

See the specifications [3] and [4] for the meaning of each field. See also [5] for other information about Server Alerted Sync implemented on Funambol server side.

To enable, or disable, the plug-in in order to listen for these particular SMS, it is necessary to register the application in a device Push Router's Registration Table. The methods used are the follows

Name	***Description***
HRESULT PushRouter_RegisterClient(LPCTSTR szContentType, LPCTSTR szAppId, LPCTSTR szPath, LPCTSTR szParams);	This method registers a client in the Push Router's Registration Table. All clients that require any interaction with the Push Router must register using this method. *szContentType*: Content-type of messages routed to the client (we use empty “”) *szAppId*: Application ID of the client (we use "x-wap-application:push.syncml") *szPath*: Path to the client's executable file (we use startsync.exe) *szParams*: Command-line parameters to be passed to the client upon launch (we use "wap_push")
PushRouter_UnRegisterClient(LPCTSTR szContentType, LPCTSTR szAppId)	This method deletes a client entry from the Registration table in the Push Router. If the client no longer wants to receive push messages from push router, the client should use this method to unregister itself. This API needs to be called once per Application ID/Content-Type combination. *szContentType*: Content-type of messages routed to the client *szAppId*: Application ID of the client

The methods are called invoking the startsync.exe program with proper command line:

```
startsync.exe register        ; it register the application
```

```
startsync.exe deregister      ; it unregister the application
```

When the SMS comes on the device and it is caught, it invokes the application registered, *szPath,* with the *szParams* command line.

```
startsync.exe wap-push
```

At this point the message is parsed and if all is correct the sync can start.

3.5. TCP/IP

This way to starts a sync is strictly related to the Notification Listener module (see 4.3). After the notlstnr dll library is configuerd and registered, the device is ready to wait for a server TCP/IP message. When this message comes on the device, due to a server modification of contacts or calendar, or a new email, the library parses the message and lauch the startsync program with the proper command line.

For more details about the Notification Listener see the 4.3. For more details about the message see the wap-push paragraph (3.4). See also [5] for other information about Server Alerted Sync implemented on Funambol server side.

4. Sync logic

This section shows the architecture of the classes and functions that are the core of the Windows Mobile Plug-in.

4.1. Client dll

The client dll is the library that contains functions to implement the *synchronization* logic of every data source (**Main** block), to set the *configuration* parameters on the device (**Settings** block), to send the sync event to the User Interface (**Events** block) and to notify the device address change to the server (**Notification** block). This section explains the content of every block of Figure 1.

4.1.1. Main

The synchronization process is the procedure that aims to have a client and a server "in sync" with the same data. This is reached by exchanging the modified data (new, update and deletes) to the server and vice versa. The data the windows mobile plug-in handles can be of different types but can be grouped in 3 main categories:

- *PIM data*: these data (items) are contacts, calendars and tasks. These are part of Microsoft Pocket Outlook Object Model that provides functions to access the objects properties. For more information about POOM see [6].
- *File data*: these items (data) are files and the content is synced to the server. The Pocket PC supports also Notes that are basically files in which is possible to insert multimedia elements too. At the moment only the textual part of them can be synced.
- *Mail data*: the items (data) are mails. The plug-in syncs only mail that are in Inbox and Outbox of the custom Funambol account.

The core of the synchronization process is driven by the Funambol C++ API: they are in charge to configure the sync engine, to establish the connection to the server, to create and to parse the syncml stream and lead the sync flow (for more information about Funambol C++ API see [7]). They expose a series of interfaces, to be implemented by the client, that need to get the data to populate the message to exchange with the server. The API provide method and classes to handle the configuration too. This latter will be discussed deeply in 4.2.2.

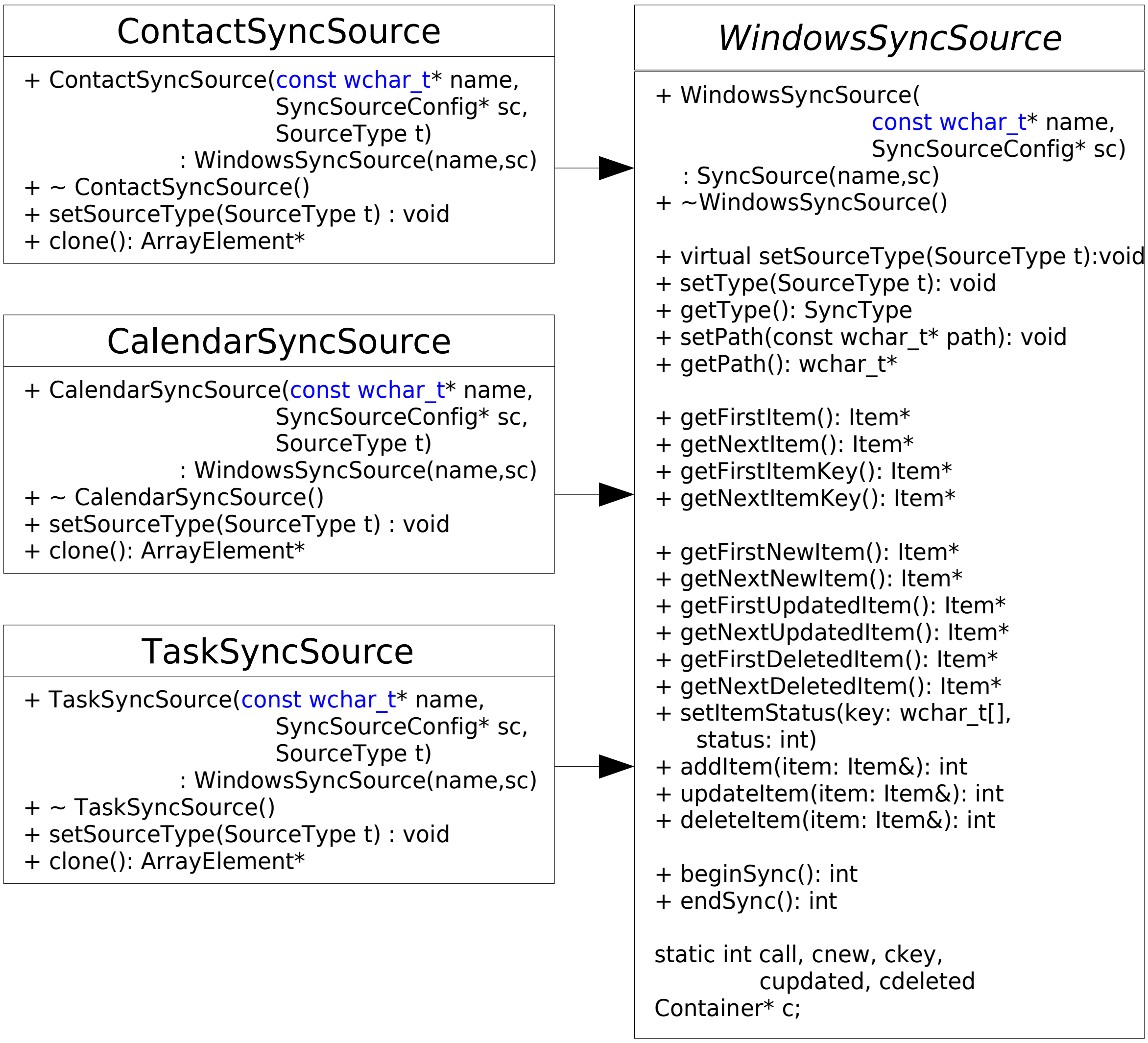
After the sync process is started and the authentication procedure is completed, the sync manager on the client side (the API SyncManager class) and the server decide which type of sync has to be performed. According to the sync type they agree, the client sync manager must fill the syncml message and send data. To populate the message it gets the data (items) asking to an interface named SyncSource. This provides methods that must be implemented by the client that allow to complete the message. At the same way, the client sync manager calls other SyncSource methods according to the server response command messages to modify the data on the client side.
Every particular data is shared between the client implementation and the C++ API in a particular object, provided by the C++ API, called SyncItem.

To handle the 3 categories of client data source, there are 3 implementation of the SyncSource interface: WindowsSyncSource for PIM data source, FileObjectSyncSource for File data source and MailSyncSource for mail data source.
The *initialize* part in Figure 1 contains also nested modules useful to communicate with the User Interface or to notify the new ip address of the device to the server. These will be analyzed later (see 4.1.2 and 4.1.3).

4.1.1.1. PIM data source

The **WindowsSyncSource** is the one to synchronize POOM items. It implements the methods to retrieve items from POOM database and to insert, modify or delete items according to Sync Manager directive. The **ContactSyncSource**, **CalendarSyncSource** and **TaskSyncSource** are specialization for the particular PIM database and represent Contact, Calendar and Task. These implementation must set a SourceType to figure out which is the POOM database to work on. The Pocket Outlook SourceType must be one of the follows:

OL_CONTACTS = 10
OL_CALENDAR = 9
OL_TASK = 13



```
ContactSyncSource
+ ContactSyncSource(const wchar_t* name,
                    SyncSourceConfig* sc,
                    SourceType t)
        : WindowsSyncSource(name,sc)
+ ~ ContactSyncSource()
+ setSourceType(SourceType t) : void
+ clone(): ArrayElement*
```

```
CalendarSyncSource
+ CalendarSyncSource(const wchar_t* name,
                     SyncSourceConfig* sc,
                     SourceType t)
        : WindowsSyncSource(name,sc)
+ ~ CalendarSyncSource()
+ setSourceType(SourceType t) : void
+ clone(): ArrayElement*
```

```
TaskSyncSource
+ TaskSyncSource(const wchar_t* name,
                 SyncSourceConfig* sc,
                 SourceType t)
        : WindowsSyncSource(name,sc)
+ ~ TaskSyncSource()
+ setSourceType(SourceType t) : void
+ clone(): ArrayElement*
```

```
WindowsSyncSource
+ WindowsSyncSource(
                const wchar_t* name,
                SyncSourceConfig* sc)
   : SyncSource(name,sc)
+ ~WindowsSyncSource()

+ virtual setSourceType(SourceType t):void
+ setType(SourceType t): void
+ getType(): SyncType
+ setPath(const wchar_t* path): void
+ getPath(): wchar_t*

+ getFirstItem(): Item*
+ getNextItem(): Item*
+ getFirstItemKey(): Item*
+ getNextItemKey(): Item*

+ getFirstNewItem(): Item*
+ getNextNewItem(): Item*
+ getFirstUpdatedItem(): Item*
+ getNextUpdatedItem(): Item*
+ getFirstDeletedItem(): Item*
+ getNextDeletedItem(): Item*
+ setItemStatus(key: wchar_t[],
   status: int)
+ addItem(item: Item&): int
+ updateItem(item: Item&): int
+ deleteItem(item: Item&): int

+ beginSync(): int
+ endSync(): int

static int call, cnew, ckey,
           cupdated, cdeleted
Container* c;
```

Figure 8: WindowsSyncSources and derived sync sources class diagram

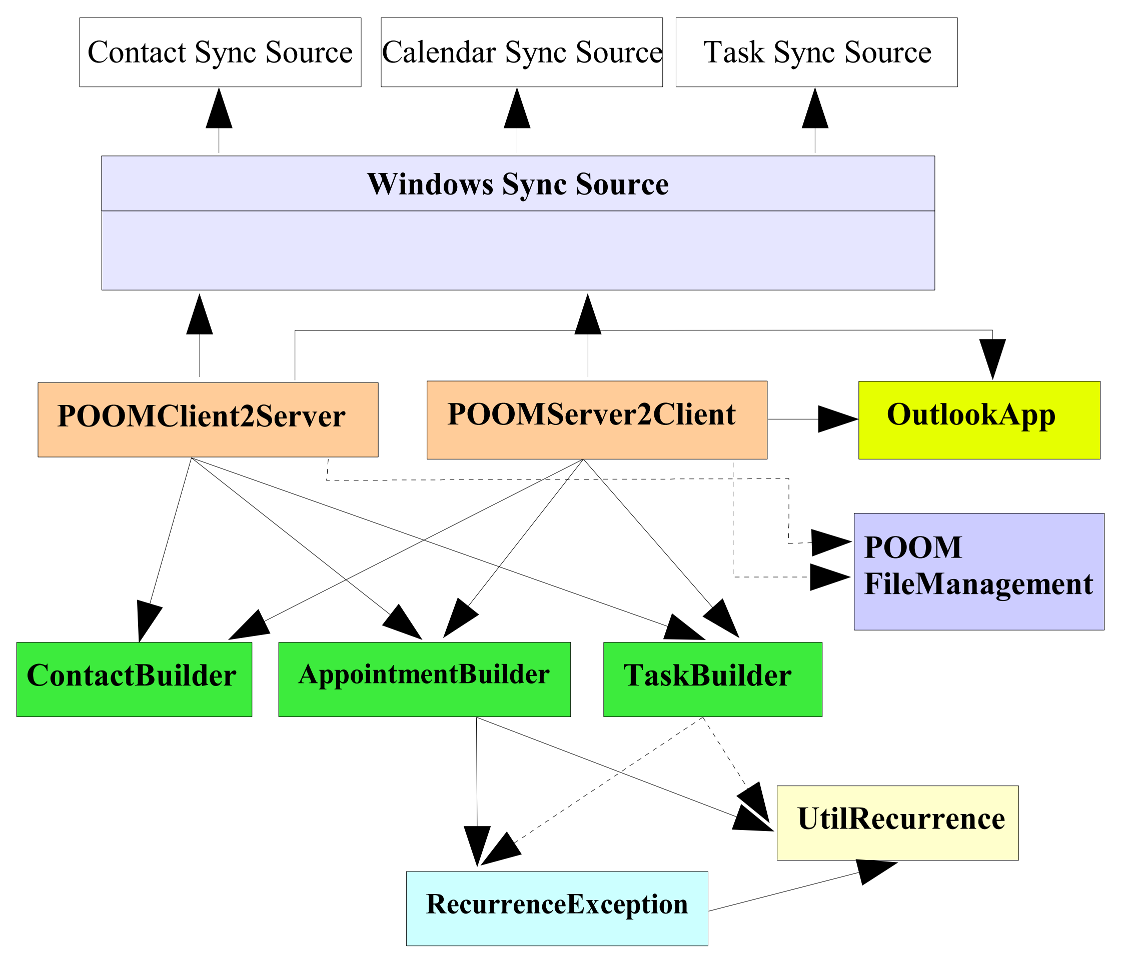


Figure 9: POOM Sync Sources class iteration

To get all the informations about every modified items, the SyncSources methods use several functions collected in different modules.
The most important functions in the modules are described below.

- **POOMClient2Server:** collection of functions to handle the contact, calendar and task items that have to be exchanged from client to server.

Name	Description
void setAllItemsPOOM(Container* c, int dataType, wchar_t* path)	Check all the items of the dataType source to set in the Container::allItems array. There are only the name of the items.
void setModifiedItemsPOOM(Container* c, int dataType, wchar_t* path)	Check the items new, modified and deleted of the dataType source to set in the Container::newItems, Container::updatedItems and Container::deletedItems arrays. In the arrays there are the complete Items (name and content)
int deleteAllItems(int dataType)	It deletes all the items of the dataTyoe source. Used for the “replace local data” sync mode.
void fillSyncItem(SyncItem* syncItem, int dataType)	It fills the SyncItem object (SyncItem::data using the key of the SyncItem.

- **POOMServer2Client**: collection of functions to handle the contact, calendar and task items that have to be exchanged from server to client:

Name	Description
long manageNewItems(SyncItem* item, int dataType, long *oid, wchar_t* path)	It handles a new item of dataType and store it in the database. The *oid is the key of the just created element.
long manageUpdatedItems(SyncItem* item, int dataType, wchar_t* path)	It updates the item of dataType with the data in the SyncItem
long manageDeletedItems(SyncItem* syncItem, int dataType)	It deletes the item of dataType

- **POOMFileManagement**: collection of functions to handle the interaction with the file system. They are used to read and write the cache of every data source before and after a sync process. Considering for instance the Contact data source. The *cache* is a file generated when a sync is completed and contains the ID of all the contacts on the device and its corresponding crc computation. The cache is used at the next sync to compare and understand which are the new, modified and deleted contacts since the last sync.

Name	Description
void readFromFile(long* previousCountItem, vector<long> &previousOid, vector<long> &previousHash, int dataType, wchar_t* path)	It reads the cache file of dataType source and fills the vector with the previous OID and the corresponding crc value
void writeToFile(vector<long> &currentOid, vector<long> &currentHash, int dataType, wchar_t* path) {	It writes the cache file of dataType source with the current OID and its corresponding crc value.

- **ContactBuilder**: collection of functions to handle a single contact item. It provides methods to format a IContact Pocket Outlook object in *SIF-C or vCard* structure and methods to parse a SIF-C or vCard structure to set a IContact Pocket Outlook object. The SIF format is a representation of the contact item using the Syncj4 Interchange Format, an XML propertyName/propertyValue structure, mostly used between Funambol client and server. To understand more about SIF see [8].

Name	Description
void populateContactStringItem(wstring &contactStringItem, IContact *pContact)	It gets every property name and value from the IContact object and build the SIF XML structure
void completeContact(IContact *pContact, wchar_t * ptrData)	It parses the SIF XML structure and fills every property of the IContact object
VObject* ContactToVObject(IContact *pContact)	It gets every property name and value from the IContact object and build the VObject structure
void VObjectToContact(IContact *pContact, VObject *vo)	It parses the VObject structure and fills every property of the IContact object

- **AppointmentBuilder**: collection of functions to handle a single calendar item. It provides methods to format an IAppointment Pocket Outlook object in *SIF-E* or *iCalendar* structure and methods to parse a SIF-E or iCalendar structure to set an IAppointment Pocket Outlook object.

Name	Description
void populateAppointmentStringItem(std::wstring &appointmentStringItem, IAppointment *pAppointment, BOOL isCRC)	It gets every property name and value from the IAppointment object and build the SIF XML structure. IsCrc value is used to discriminate if the structure is used to compute the crc or not.
void completeAppointment(IAppointment *pAppointment, wchar_t *ptrData, IPOutlookApp* polApp, wchar_t* path)	It parses the SIF XML structure and fills every property of the IAppointment object
VObject* AppointmentToVObject(IAppointment *pAppointment)	It gets every property name and value from the IAppointment object and build the VObject structure
void VObjectToAppointment(IAppointment *pAppointment, VObject *vo)	It parses the VObject structure and fills every property of the IAppointment object

- **TaskBuilder**: collection of functions to handle a single task item. It provides methods to format an ITask Pocket Outlook object in *SIF-T* or *iCalendar (currently not supported)* structure and methods to parse a SIF-T (or iCalendar) structure to set an ITask Pocket Outlook object.

Name	Description
void populateTaskStringItem (std::wstring &taskStringItem, ITask *pTask, BOOL isCRC)	It gets every property name and value from the ITask object and build the SIF XML structure. IsCrc value is used to discriminate if the structure is used to compute the crc or not.
void completeTask (ITask *pTask, wchar_t * ptrData)	It parses the SIF XML structure and fills every property of the ITask object
VObject* TaskToVObject(ITask *pTask)	It gets every property name and value from the ITask object and build the VObject structure
void VObjectToTask(ITask *pTask, VObject *vo)	It parses the VObject structure and fills every property of the ITask object

- **UtilsRecurrence**: collection of functions to handle the Calendar and Task Recurrences. It is used also for the Calendar Exceptions too.

Name	Description
wstring getRecurrenceTags(IRecurrencePattern* pRecurrence, VARIANT_BOOL isRecurring, int x, int y, OlDefaultFolders olFolder, DATE startdate, BOOL isAllDay)	Return the SIF structure of the recurrence appointment. Used for calendar and tasks
wstring createExceptions(IRecurrencePattern* pRecurrence, DATE startdate, BOOL isAllday)	Create the exceptions SIF structure if exists. Used on calendar only.
wstring createException(IException* pException, DATE startdate, BOOL isAllday)	Create the single exception SIF tag. Used on calendar only.
void parseRecurrenceTags(IRecurrencePattern *pRecurrence, const wchar_t* ptrData, DATE recurrenceStart, int x, int y)	Parse the Recurrence SIF structured that comes from the server
void parseExceptionTags(IRecurrencePattern* pRecurrence, IAppointment* pAppointment, const wchar_t* ptrData, IPOutlookApp* polApp, const wchar_t* path);	Parse the Exception SIF structured that comes from the server

- **RecurrenceException**: It's a class container that stores all the fields needed to handle a Calendar Recurrence Exception. A Calendar Exception is a particular object generated by the POOM Manager when the user modifies or deletes an existing occurrence of a recurrent event. Handling the exceptions on the Windows Mobile devices has to be consistent with the exceptions handled by other clients so they are managed in a particular way. More at Appendix B.

Name	Description
RecurrenceException()	Constructor
~RecurrenceException()	Destructor
set/getOriginalDate(DATE date)	set/get the Original Date the appointment occurs
set/getStart(DATE date)	set/get the start date for the exception event
set/getEnd(DATE date)	set/get the end date for the exception event
set/getSubject(wstring s)	set/get the subject for the exception event
set/getLocation(wstring s)	set/get the location for the exception event
set/getBody(wstring s)	set/get the body for the exception event
set/getAllDayEvent(BOOL s)	set/get the all day event for the exception event
set/getIsDeleted(BOOL s)	set/get the delete property for the exception event
set/getBusyStatus(long s)	set/get the busy status for the exception event
isOriginalEqualsStart()	Check if the start date is equal of original

- **OutlookApp**: singleton class used to obtain and handle the connection with the Pocket Outlook application. It connects with the POOM data store manager and keep the handle to be shared with all the methods that need to ask the PIM items.

Name	Description
static OutlookApp* getInstance()	Create the object instance
static void dispose()	Delete the object instance
IPOutlookApp* getPolApp()	Get the IPOutlookApp* POOM object. It is the application manager to interact with the PIM data

	store

4.1.1.2. File data source

The **FileObjectSyncSource** is the one to synchronize items that are basically files. It implements the methods to get file items and to create, modify or delete items according to server commands. The **BriefcaseSyncSource** and **NoteSyncSource** must set a SourceType to figure out which is the files to work on. The file SourceType could be one of the follows:

OL_BRIEFCASE = 600
OL_NOTES = 601

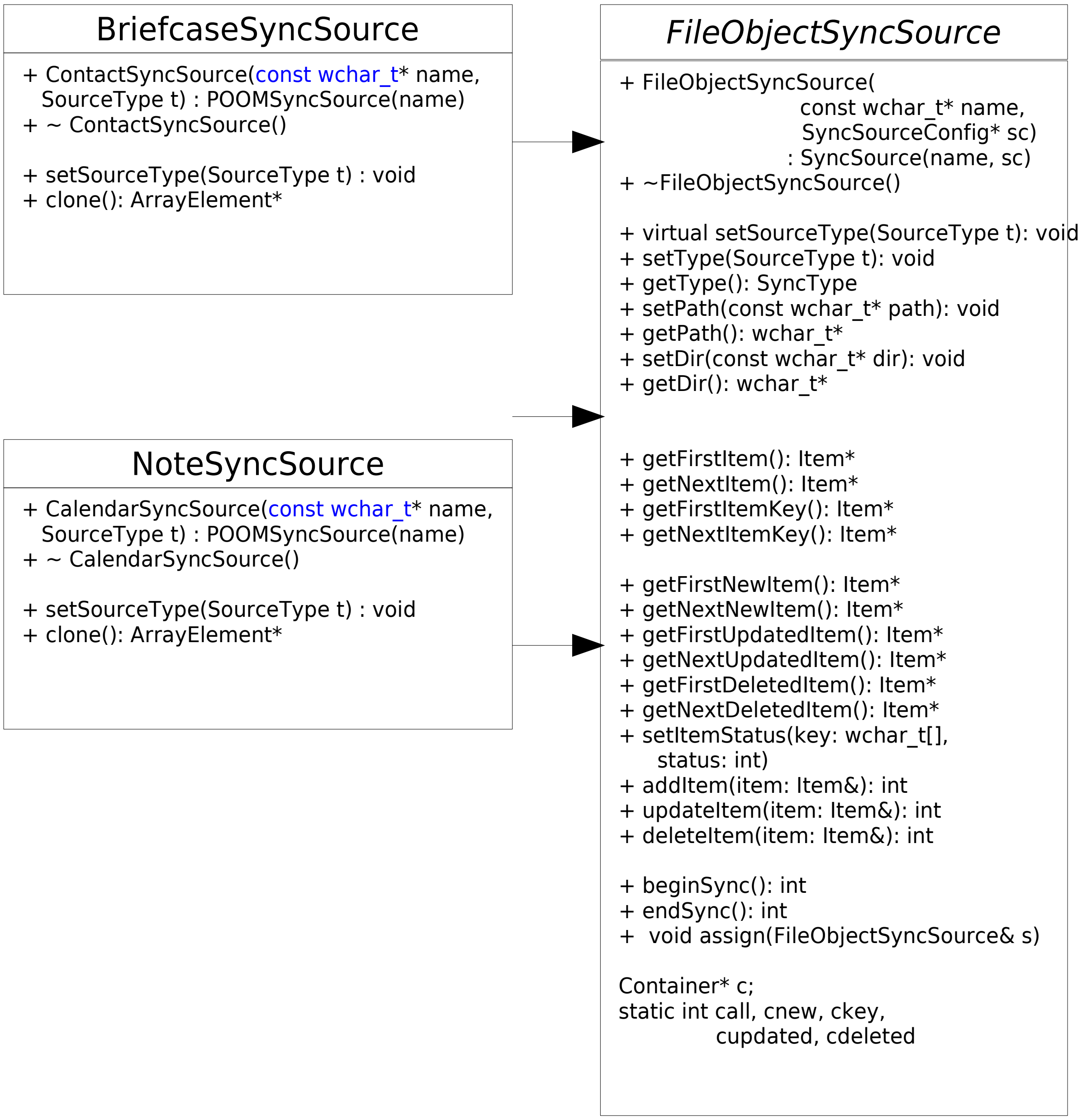

BriefcaseSyncSource
+ ContactSyncSource(const wchar_t* name,
SourceType t) : POOMSyncSource(name)
+ ~ ContactSyncSource()
+ setSourceType(SourceType t) : void
+ clone(): ArrayElement*
NoteSyncSource
+ CalendarSyncSource(const wchar_t* name,
SourceType t) : POOMSyncSource(name)
+ ~ CalendarSyncSource()
+ setSourceType(SourceType t) : void
+ clone(): ArrayElement*
FileObjectSyncSource
+ FileObjectSyncSource(
const wchar_t* name,
SyncSourceConfig* sc)
: SyncSource(name, sc)
+ ~FileObjectSyncSource()
+ virtual setSourceType(SourceType t): void
+ setType(SourceType t): void
+ getType(): SyncType
+ setPath(const wchar_t* path): void
+ getPath(): wchar_t*
+ setDir(const wchar_t* dir): void
+ getDir(): wchar_t*
+ getFirstItem(): Item*
+ getNextItem(): Item*
+ getFirstItemKey(): Item*
+ getNextItemKey(): Item*
+ getFirstNewItem(): Item*
+ getNextNewItem(): Item*
+ getFirstUpdatedItem(): Item*
+ getNextUpdatedItem(): Item*
+ getFirstDeletedItem(): Item*
+ getNextDeletedItem(): Item*
+ setItemStatus(key: wchar_t[],
status: int)
+ addItem(item: Item&): int
+ updateItem(item: Item&): int
+ deleteItem(item: Item&): int
+ beginSync(): int
+ endSync(): int
+ void assign(FileObjectSyncSource& s)
Container* c;
static int call, cnew, ckey,
cupdated, cdeleted

Figure 10: FileObjectSyncSources and derived sync sources class diagram

To get all the information about every modified items, the SyncSources methods use several functions collected in different modules.

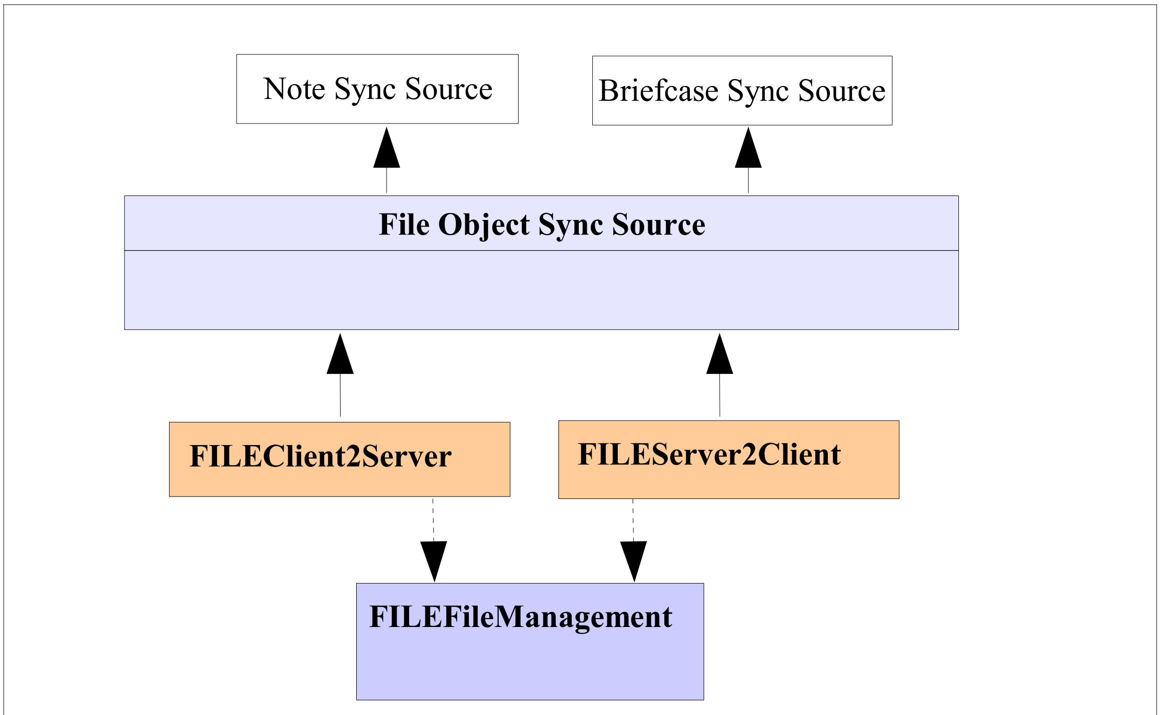


Figure 11: File Sync Source class iteration

The most important functions in the modules are described below.

- **FILEClient2Server**: collection of functions to handle the file items to be sent to the server. It provides methods to get all modified file items and format in a structure, provided by the C++ API ([7]), named FileData. On the Windows Mobile Pocket PC, also Notes are represented as files. Therefore, for Pocket PC version only, it is possible to handled notes as well. Actually, only the textual part of the notes are properly treated. The notes are exchanged in a XML *SIF-N* structure and the following methods are able to handle them properly.

Name	Description
void setAllItemsFILE(Container* c, int dataType, const wchar_t* path, const wchar_t* dir)	Set all the items to sync of *dataType* source, that are in the *dir* directory, in the Containter object.
void setModifiedItemsFILE(Container* c, int dataType, const wchar_t* path, const wchar_t* dir)	Set all the modified items to sync of *dataType* source, that are in the *dir* directory, in the Containter object.
void setAllItemsFILEKey(Container* c, int dataType, const wchar_t* path, const wchar_t* dir);	Set all the keys (the name of the files) in the Container object. This function is used by the getFirstItemKey method to delete all the device item before a replace from server sync.

- **FILEServer2Client**: collection of functions to handle the file items sent by the server. It provides methods to manage both FileData format or no structured data. “No structured data” means the server sends the content of the file directly in the syncML message without packaging in the appropriate FileData structure. The function keeps this behavior to maintain compatibility with old servers. For the Windows Mobile Pocket PC version, in which Notes are exchanged in a XML *SIF-N* structure, the following methods are able to handle this format properly.

Name	Description
wchar_t* FILEmanageNewItems(SyncItem* item, int dataType, const wchar_t* path, const wchar_t* dir)	Store the item of *dataType* source, that is in the *dir* directory, and return the file name
long FILEmanageUpdatedItems(SyncItem* item,	Update the item of *dataType* source, that are in

int dataType, const wchar_t* path, const wchar_t* dir)	the *dir* directory. It returns always 0.
long FILEmanageDeletedItems(SyncItem* item, int dataType, const wchar_t* path, const wchar_t* dir)	Delete the item of *dataType* source, that are in the *dir* directory. It returns always 0.

- **FileFileManagement**: collection of functions to read/write in the file system. It provides several useful methods to read or write the content of the files to be packaged by the previous FILEClient2Server and FILEServer2Client modules. It also have the functions to read and write the cache file. Follows some samples:

Name	**Description**
int readFilenameFromFile(int dataType, const wchar_t* path, wchar_t*** ptrArrayFilename)	Read the file name from the cache of *dataType* and fills the ptrArrayFilename.
void writeCurrentFileItems(int dataType, const wchar_t* path, const wchar_t* dir);	Write the cache of *dataType* whit the filename that are in the *dir* directory.
void readByteFromFile(wchar_t* fName, byte *ptr, int* numBytes);	Read a file returning the ptr to the byte read and the number of byte read
void writeByteToFile (wchar_t* fName, byte *ptr, int numBytes);	Read numByte byte of ptr in the fName file
void encodeKey(wchar_t* key);	Encode in b64 the key (filename) passed. It needs to avoid particular char in the filename
void decodeKey(wchar_t* key);	Decode from b64 the key (filename)

4.1.1.3. Mail data source

The **MailSyncSource** is the one to synchronize items that are emails. It implements all the common methods to get email items and to create, modify or delete items according to server commands.
The MailSyncSource uses some classes provided by the C++ API that represent a mail object. These classes have methods to parse a syncML mail representation and to format a mail object into syncML. The C++ API classes (EmailData, MailMessage and FolderData) are stored into a MailClientData class that is used to share the mail information across the methods that read and write in the device mail data store.

```
MailSyncSource

+ MailSyncSource(const wchar_t* name, SyncSourceConfig* sc);
+ MailSyncSource(MailSyncSource &s);
+ ~MailSyncSource();

+ setPath(const wchar_t* p): void
+ getPath(): const wchar_t*
+ setFolderToSync(const wchar_t* t): void
+ getFolderToSync(): const wchar_t*
+ isInFolderToSync(const wchar_t* toCheck): bool

+ setIsMailInOutbox(bool value): void
+ getIsMailInOutbox(): bool
+ setIsMailInInbox(bool value): void
+ getIsMailInInbox(): bool
+ setFailedSendMailInOutbox(bool value): void
+ getFailedSendMailInOutbox(): bool
+ setMaxMailMessageSize(int value): void
+ getMaxMailMessageSize(): int
+ setIsSyncInclusive(bool value): void
+ getIsSyncInclusive(): bool

+ getFirstItem(): Item*
+ getNextItem(): Item*
+ getFirstItemKey(): Item*
+ getNextItemKey(): Item*

+ getFirstNewItem(): Item*
+ getNextNewItem(): Item*
+ getFirstUpdatedItem(): Item*
+ getNextUpdatedItem(): Item*
+ getFirstDeletedItem(): Item*
+ getNextDeletedItem(): Item*
+ setItemStatus(key: wchar_t[],
    status: int)
+ addItem(item: Item&): int
+ updateItem(item: Item&): int
+ deleteItem(item: Item&): int

+ beginSync(): int
+ endSync(): int
+  void assign(FileObjectSyncSource& s)

- MailClient2Server *mailClient2Server
- MailServer2Client *mailServer2Client;

- SyncItem *buildSyncItem(MailClientData* m, ArrayList& extras)
- bool setMailData(SyncItem &item, const wchar_t *parent,
                       MailClientData &m, bool isUpdate)
- bool setFolderData(SyncItem &item, FolderData &folder)
```

Figure 12: MailSyncSource class diagram

Follows an explanation of some important functions and members of the mail sync source.

Name	***Description***
MailClient2Server* mailClient2Server	Pointer to the object responsible to retrieve the mail information from the mail datastore. See later for more about it
MailServer2Client* mailServer2Client	Pointer to the object responsible to store the mail server information to the mail data store. See later for more about it
SyncItem* buildSyncItem(MailClientData* m, ArrayList& extras)	Static function that formats a retrieved mail object from the device data store into a syncML item.

bool setMailData(SyncItem &item, const wchar_t* parent, MailClientData &m, bool isUpdate)	Static function that parses a syncML mail item and fill a MailClientData object to be used in the mail data store.
static bool setFolderData(SyncItem &item, FolderData &folder)	Static function that parses a syncML mail FolderData object.
set/getFolderToSync	Set/Get the mail folder to sync. They could be I=Inbox, O=Outbox. Other main folders (S=Sent, T=Trash, D=Draft) are no more synced
set/getIsMailInOutbox	Set/Get if at the end of the sync there is a mail in Outbox folder. Used in previous version when needed a double sync to send the email. (TBR)
set/getIsMailInInbox	Set/Get if there is a new mail in inbox. Used the understand if play the sound.)TBR because the default sound will be used).
Set/getFailedSendMailInOutbox	Set/Get if there was a problem in sending the email (on client side or server side)
set/getIsSyncInclusive	Set/Get if the current sync is performed with an inclusive filter.
set/getMaxMailMessageSize	Set/Get the max message size supported by the application. It is get from the configuration.

To get all the information about every modified items, the SyncSource methods use several functions collected in different classes.

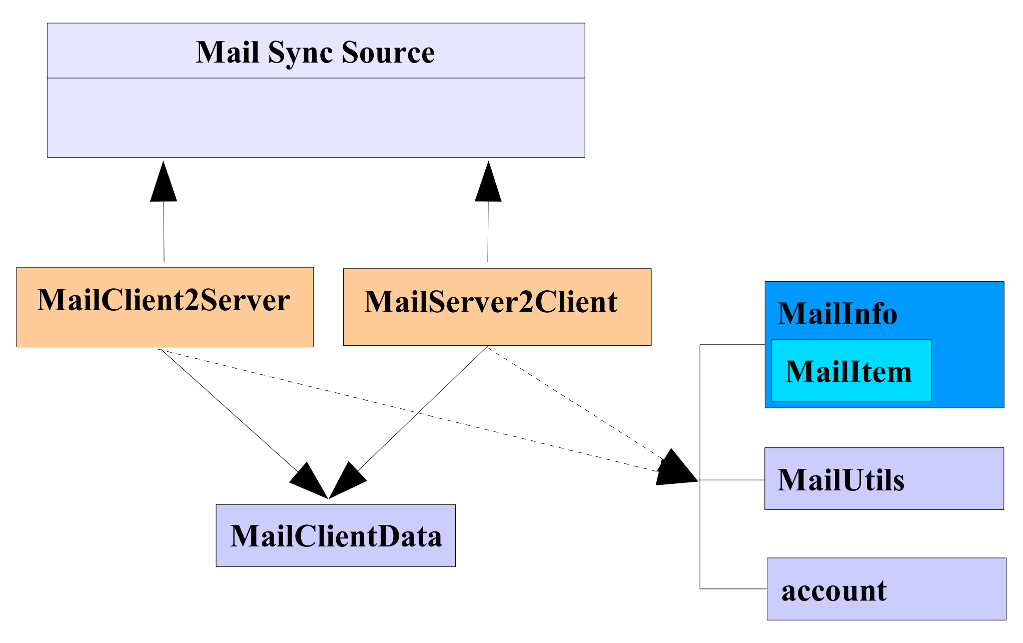


Figure 13: Mail Sync Source class iteration

- **MailClient2Server**: class containing functions to retrieve the item to send to the server. It provides methods to get all the mails or only the new, modified and deleted mails according to the sync to perform. It has methods to log in the Mail API database and to read all the properties of a mail object. The most important functions are:

Name	Description
IMAPISession* logOn()	Log in the MAPI database
void logOff(IMAPISession* pSession);	Log off the MAPI database

MailClientData* completeObject(IMessage* pmsg, wchar_t folder, BOOL isUpdate, BOOL keyOnly = FALSE);	Create a MailClientData object containing a MailMessage object that is filled with all the mail properties (Sender, subject, To, attachments, body...). It fills only the mail flag (read flag) if the client is sending a mail update.
HRESULT setSync (int mode, const wchar_t* path, ArrayList& extra);	Based on the *mode (slow, two-way, one-way...),* the method calls *GetAllMessages* or *GetModificatedMessages* functions that prepare all the items to be exchanged
HRESULT GetAllMessages(ENTRYLIST& entryList, LPMAPIFOLDER m_pInBoxFolder, const wchar_t* path, wchar_t folderIdentification)	Fill a full list (*entryList)* with all the mail items that are specified in the *m_pInBoxFolder.*
HRESULT GetModificatedMessages (ENTRYLIST& newMessages, ENTRYLIST& updatedMessages, ArrayList** deletedMessages, LPMAPIFOLDER m_pInBoxFolder, const wchar_t* path, wchar_t folderIdentification, ArrayList& extra);	Fill lists of new, update and deleted mail items of a specified mail folder. ArrayList& extra (TBR)
MailInfo* readPreviousMail(const wchar_t* path, wchar_t folderId, ArrayList& extra);	Return a MailInfo object of a specified mail folder from the previous cache. It needs to check for the new, updated, modified items from the last sync (TBR extra)
void writeCurrentMails(const wchar_t* path, ArrayList& extra);	Write the current mail id item cache for every folder
HRESULT moveMessageFromOutboxToSentItemsFolder(const wchar_t* key);	After a message is sent successfully be the server the mail in Outbox folder is moved to the Sent one

- **MailServer2Client**: class containing functions to store the mail item into the device data base. It provides methods to add a new mail, to modify or delete an existing one. The most important functions are:

Name	***Description***
wchar_t* addMessage(MailClientData* m);	Wrapper function used by the MailSyncSource to add a new mail item.
HRESULT updateMessage(MailClientData* m);	Wrapper function used by the MailSyncSource to modify an update mail item.
HRESULT deleteMessage(MailClientData* m);	Wrapper function used by the MailSyncSource to delete a mail item.
HRESULT SetMessageProps(IMAPISession* pSession, LPMESSAGE pmsg, MailClientData* m, BOOL onlyFlags = false);	Function that reads all the mail fields from the MailClientData (Subject, To, body, attachments...) and stores it in the device mail datastore. If onlyFlags is true only the flag (i.e. Read flag) are updated.

- **MailClientData**: class containing the C++ API EmailData object, that is a representation of the syncML email object. The EmailData has methods to format itself to a syncML and to parse a syncML structure. Through this class is possible to access every field of the email message. The MailClientData is used by the MailSyncSource, the MailClient2Server and the MailServer2Client to easily handle the mail message. The most important members are the following (note they have their own set/get methods):

Name	***Description***
EmailData* emailData	The C++ API object that represents a mail message. It provides methods to read/write all the mail message properties (Subject, To, attachments...) and the mail flags (read, forwarded..)
wchar_t* entryId	The id of the mail in the format I/AAABBCC00954. It is the folder in which the mail will be stored or from it is retrieved / the b64 conversion of the binary id of the mail message

wchar_t* folder	The folder of the mail (I=Inbox, O=Outbox, S=Sent, T=Trash, D=Draft)
int maxMailMessageSize	The max mail message supported by the client.
int currentMessageSizeFilter	The current filter mail size
bool isSyncInclusive	It is true if the current sync process is with an inclusive filter.

- **MailInfo**: class used to represent the structure of the cache of the last sync. The cache is a file for every folder to sync (currently only Inbox. Outbox doesn't need any cache file because it is always empty after a sync. If not empty, every mail in this folder must be sent again).
  The cache mail file has a structure like

```
<mails>
 <num>1</num>
 <last>39126.782743</last>
 <lastTime>20070213T174709Z</lastTime>
 <item>
    <id>I/AABBCC0004345ED</id>
    <read>true</read>
  </item>
</mails>
```

Name	Description
DATE last;	The last time the sync was succesfully completed
int num	The number of item structure the file contains
ArrayList* mailItems	List of all the MailItem object representing a single email object

- **MailItem**: class used to represent the id and the *read* flag state of the mail at the previous sync.

Name	Description
char* id;	The id of the mail that there was at the end of the last sync
bool read;	Flag to say if the mail was already read of not

- **MailUtils**: collection of utility functions used by the mail classes

Name	Description
char* convertBinaryToChar(SBinary sbEntry, wchar_t folder)	Convert the binary internal mail ID into a char string. It uses the b64 conversion and an element to understand the folder in which the mail is. The result is something like I/AABBCC00034543
void getFolderToSync(MailSyncSourceConfig &sc, OUT wchar_t* tt);	Get the folder to sync from the plug-in configuration
char* createMessageID(const wchar_t* entryID)	Create the MessageID header in the outgoing mail if it doesn't have any.

- **account**: collection of functions related to the Funambol custom email account.

Name	Description
HRESULT modifyAccount(const wchar_t* name, const wchar_t* replyAddress)	Used to modify the Funambol account setting the visible name and the mail address
HRESULT getAccountInfo(wchar_t* name, wchar_t* replyaddress)	Get the name and the reply address from the Funambol mail account
BOOL doesFunambolAccountExist()	Check if the Funambol mail account exists. Used by the UI methods to decide if to show the mail settings.

4.1.1.4. Initialize

This section describes the functions needed to create the sync engine and begin the sync process. These methods are grouped in two principal modules: the *CustomSyncClient* and the *maincpp*.
The C++ API provides a SyncClient interface and its implementation that has to be instantiate to run a synchronization. This SyncClient can be extended to add specific custom functionalities: in the plug-in the customized CustomSyncClient needs to add some extra check.
More, the *initialize* module contains the entry point function that instantiate the CustomSyncClient and kickoff the synchronization process. As for the Figure 1, all of the sync runners starts the sync calling the “startsync” program (see 4.2) and this last one calls the *synchronize()* function, in the initialize block, passing the proper parameters. In the *synchronize()* function, all the needed stuffs are prepared, the SyncSource objects, their configurations, the listener objects and then the CustomSyncClient.sync() method runs finally the sync. At the end the results are analyzed and sent to the UI. The most important methods in the modules are described below.

- **CustomSyncClient**: class derived from the SyncClient C++ API base class. It is responsible to instantiare the sync engine and drive the sync process. It uses the base default methods except for the following. It is used to notify the user that the server has requested a sync for all the data of the selected source.

Name	*Description*
int continueAfterPrepareSync()	The implementation asks the user to be sure to continue because the server has requested a sync that exchange all items and can request a lot of time

- **maincpp**: module containing the entry point function that prepare all the needed things to kicks off all the sync process. It contains other useful functions to check the sync result too.

Name	*Description*
DWORD WINAPI synchronize (const wchar_t* path, const wchar_t** sources, const wchar_t** ids, const char* mode)	The main function used to start the sync process. *path* is the install path of the application. *sources* is a NULL terminated array of strings containing the source name to synchronize; *ids* is a NULL terminated array of strings containing the id of the mail that must be synced with a INCLUSIVE filter. *mode* is the sync mode used for the sync if imposed by the client
DWORD checkStartSync()	Used by the UI modules to check if there is a sync currently in progress
int startProgram(const wchar_t *app, const wchar_t *cmdline)	Used to start an *app.exe* program with *cmdline* arguments
static wstring getMailUnsent(MailSyncSource& mailSource)	Analyzes the server message if a mail cannot be sent and show a user friendly message to the user.

4.1.2. Events

In the client dll there is a group of classes that implement the SyncEvent interfaces described in C++ API Design Document ([7]). The SyncEvent interfaces are fired by the C++ API in some strategical point of the sync flow and can help the client to have feedback about the sync process.

Basically the event plug-in classes are used to notify the User Interface regarding the action the synchronization process is performing. Through the notification event it is possible to understand which data source is syncing, in which phase the sync is (the authentication phase or the receiving data from the server), or how many items the client or server is sending in order to give to the user the feeling about the sync process. Only the most important sync events are implemented.

- **SyncListenerClient**: notification methods about the main sync process

Name	Description
void syncBegin(SyncEvent &event)	Used to notify the UI the sync is started
void syncEnd(SyncEvent &event)	Used to notify the UI the sync is ended

– **SyncSourceListenerClient**: notification methods about the specific data source. The method syncSourceTotalClientItems is not impemented because the total client items to sync are notify to the UI after the procedure in which the client has discovered its modification items. For example the client contacts to be exchanged are noified to the UI in the WindowsSyncSource class (getFirstItem, getFirstNewItem)

Name	Description
virtual void syncSourceBegin(SyncSourceEvent &event)	Used to notify the UI a specific data source starts its sync
virtual void syncSourceEnd(SyncSourceEvent &event)	Used to notify the UI a specific data source ends its sync
virtual void syncSourceSyncModeRequested (SyncSourceEvent& event)	Used to notify the UI the current data source must perform a server requested sync mode (usually slow-sync if the server doesn't agree which one the client was starting)
void syncSourceTotalClientItems (SyncSourceEvent& event)	Not implemented.
void syncSourceTotalServerItems (SyncSourceEvent& event)	Used to notify the UI how many items the server wants synchronize

– **SyncItemListenerClient**: notification methods about the every exchanged item of a data source. The methods itemxxxByClient sends a notification to the UI that an item was added, modified or deleted. Currently the UI has a counter, to show to the user, that indicates the client is sending an item without discriminate if it is new, updated or deleted. The itemxxxByServer notify the UI at the same way but regarding the server items. A different handling is for the mail item. In this case the MailSyncSource is responsible to notify the UI if a mail is properly inserted, updated or deleted. Otherwise, if an error occurs or the item is a FolderData data type and not a EmailData type, the counter is not incremented.

Name	Description
void itemAddedByClient(SyncItemEvent &event)	Notify if a new item is sent to the server
void itemAddedByServer(SyncItemEvent &event)	Notify if a new item is sent by the server
void itemDeletedByClient(SyncItemEvent &event)	Notify if a deleted item is sent to the server
void itemDeletedByServer(SyncItemEvent &event)	Notify if a deleted item is sent by the server
void itemUpdatedByClient(SyncItemEvent &event)	Notify if an updated item is sent to the server
void itemDeletedByServer(SyncItemEvent &event)	Notify if an updated item is sent by the server

– **HwndFunctions**: collection of static function used to manage the handles of the User Interface windows to address the notification messages.

Name	Description
static HWND getWindowHandle()	Return the handle of the “Funambol plug-in” main window to address the messages
static void closePreviousMsgBox()	Close all previous message box if other are coming

4.1.3. Notification

This group of classes in the main library that is responsible to give the elements to say to the server that the ip of the device has changed. These classes are related to the notlstnr library (see 4.3). When the device ip changes, the notlstnr understands this modification and starts a particular sync process in which it communicates to the server this change.
This group of classes provide the sync source, the AddressChangeSyncSource, and other useful methods to register and de-register the notlstnr library.

- **AddressChangeSyncSource**: the simple sync source used to create only an item containing the new ip address of the device. Only the method getFirstItem() is implemented.

- **addresschange**: contains the function invoked by the notlstnr library that start the sync to notify the new address to the server. It prepare all the things needed to the sync like instantiate the AddressChangeSyncSource, create its configuration and fire the sync. It returns the appropriate error code if the notification fails.

Name	***Description***
AN_ResponseCode notifyAddressChange(const wchar_t *context)	Method that prepare all the needed to start the sync process.

- **s4n_service**: module containing function to register and de-register the notlstnr library as a service.

Name	***Description***
int registerService()	Register the notlstnr library at the appropriate Operating System process
int deregisterService()	Remove the notlstnr library from the OS process

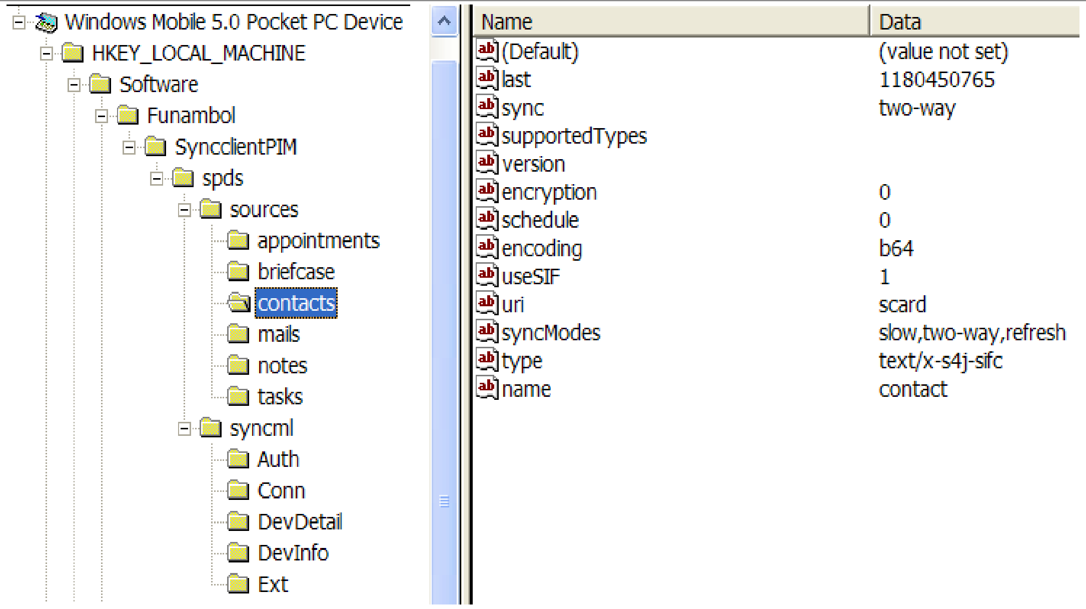
- **checknet**: module containing useful functions check if the network is active and the ip address of the device

Name	***Description***
bool IsNetPresent()	Check if a network is available
void getIPAddress(wchar_t* address)	Return the ip address

4.1.4. Settings

The client *settings* are a representation of the configuration the plug-in needs to work. Basically, they are an extension of the default provided by the C++ API. The basic configuration needs to the API to create a well formed syncml message: these information are username, password, URL, source names or their sync mode. Other extra parameters, specific for the plug-in, are needed to store some user preferences, like schedule time, notification subscription, log level or which folder use to sync files.

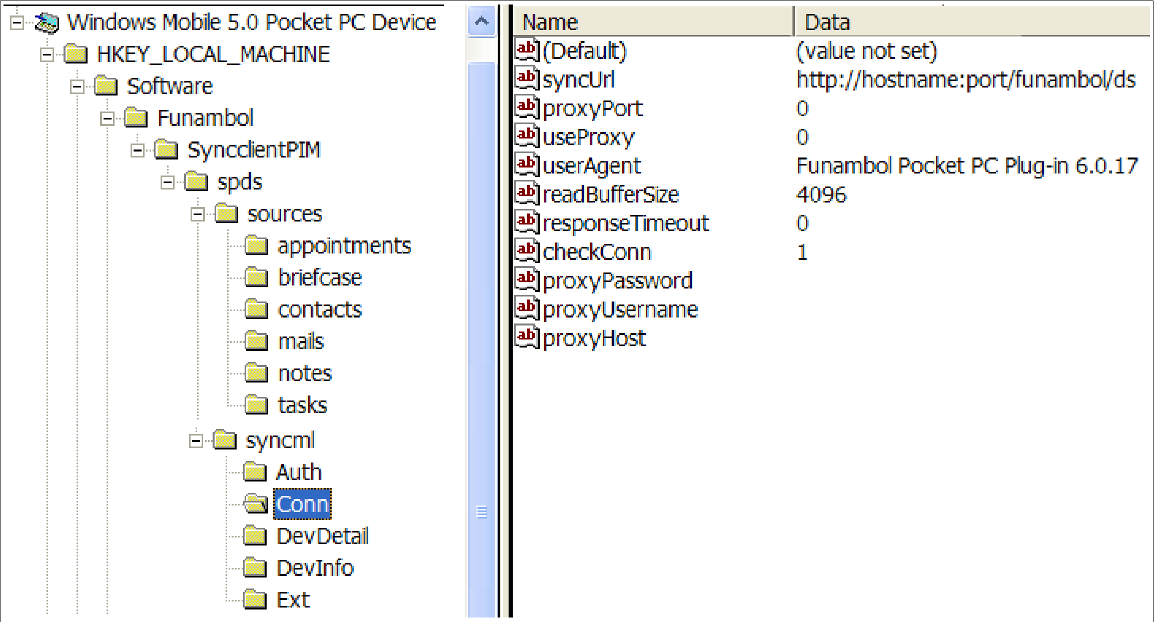
On Windows Mobile the repository in which memorize the settings is the common windows registry. During the installation procedure all the registries are filled with the default values and some of them, like server url, username and password, must be changed before make the first synchronization.
A scratch of the configuration structure of the data source to sync is as follow:



Name	Data
(Default)	(value not set)
last	1180450765
sync	two-way
supportedTypes	
version	
encryption	0
schedule	0
encoding	b64
useSIF	1
uri	scard
syncModes	slow,two-way,refresh
type	text/x-s4j-sifc
name	contact

Figure 14: Settings detail for source Contact

The node *sources* contains all the keys related to every data source configuration. Every key (contact, calendar...) contains properties that are necessary to describe the current data source and its state as if it is selected to be synced (*sync*) , the last sync time (*last*), the format of its items (*format*) and so on .



Name	Data
(Default)	(value not set)
syncUrl	http://hostname:port/funambol/ds
proxyPort	0
useProxy	0
userAgent	Funambol Pocket PC Plug-in 6.0.17
readBufferSize	4096
responseTimeout	0
checkConn	1
proxyPassword	
proxyUsername	
proxyHost	

Figure 15: Settings detail of syncml common parameters

The node *syncml* contains the keys related to the general syncML protocol that needs to the sync process. Here there are information about the server URL to sync against, the username and password, the device id and so on. To learn more about the configuration parameters see [7].

As said before, the plug-in uses these basic configuration parameters but needs to add extra properties to handle behaviors for specific purposes as the scheduler process, the TCP/IP or SMS push action, the use of the data encryption method or the way to format the item data. These values can be related to a single data source, as for the way to format the single item data, and it is stored in the *sources/datasource* branch; more particular clients properties, as push action, are stored at the *SyncclientPIM* level. A complete list of the specific plug-in extra parameters are in the follow table. Note that mail source has its own configuration class provided by the C++ API:

Sources custom properties:

Name	***Description***
All sources node	
Schedule	It is the number of minutes the corrispondent source has to start the sync
UseSIF	1 if the data source are exchanged in SIF format or 0 if it is not used. The format depends from every specifiec source
Briefcase and Note (only pocket pc version)	
Dir	The directory in which there are the files or note to sync

SyncclientPIM custom properties:

Name	***Description***
Sms	Indicates if the device has to be notified by sms (1 true, 0 false)
Push	Indicates if the device has to be notified by TCP/IP (1 true, 0 false)
PluginVersion	The install procedure writes the current installed version
Path	The path the plug-in is instsalled
CradleNotif	Indicates if the sync must start when the device is cradled (1 true, 0 false)
IsPortal	Indicates if the version installed is opensource or portal(1 portal, 0 opensource)
PushPort	The port on which the device is listening for the server notification message (4745)
SvrNotified	Indicates if the server was properly notified by the client

To handle the settings parameters, the client dll library uses *ClientSettings* class, that represents the configuration structure. It is a singleton class and it is instantiated at the beginning of the sync process by *startsync* program. Also the UI creates its own instance to load and save settings (see 3.1).

The *ClientSettings* is used by the *maincpp* module (see 4.1.4) to read all configuration needed to start the process and by the User Interface, to show to the user the current parameters and permit their changes. There is also another module, *SettingFunctions*, contains utility methods to access the registry and retrieve only specific values, without loading the entire structure.

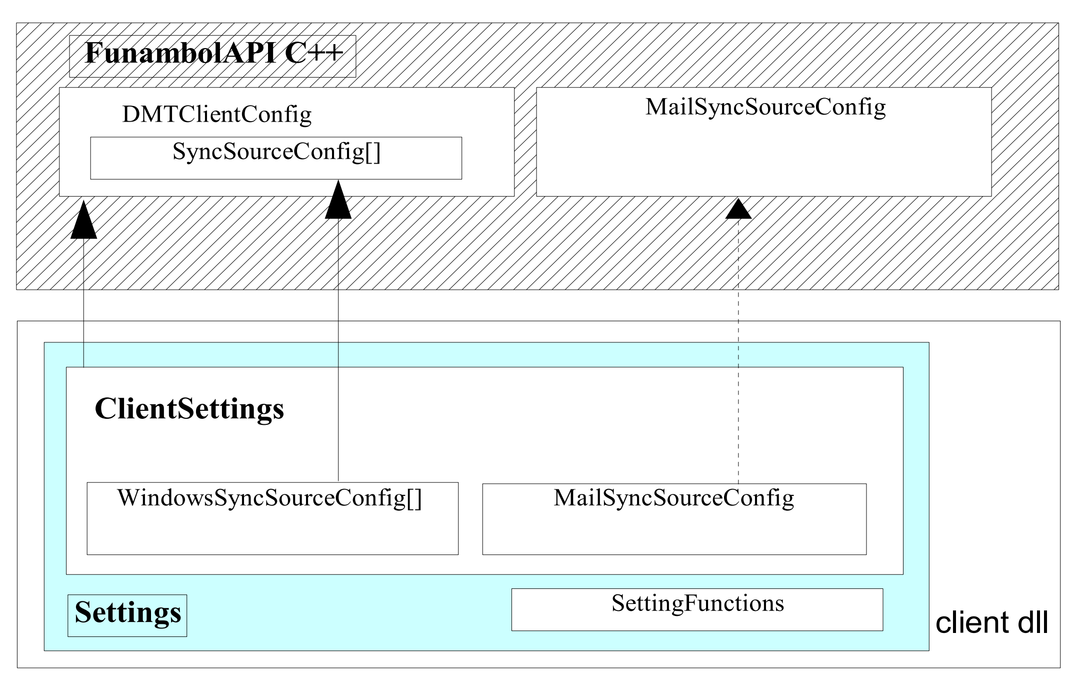


Figure 16: Plug-in settings classes details

The *ClientSettings* is a class that derives from DMTClientConfig of the API. The DMTClientConfig contains a default configuration that loads the default parameters that are common for the general syncml settings and for every sources. It contains an array of SyncSourceConfig that represent the configuration for all the data sources.

The WindowsSyncSourceConfig is a plug-in settings class that works like a proxy for the common properties of the SyncSourceConfig and handle other custom properties for every source. The schema of the **WindowsSyncSourceConfig** is like follows:

Name	***Description***
SyncSourceConfig* s;	Internal pointer to the SyncSourceConfig of the DMTClientConfig. Through this pointer, the class can get the common properties from the basic class.
string useSif	Custom property for the source
string dir	Custom property for the source
string schedule	Custom property for the source
WindowsSyncSourceConfig::WindowsSyncSourceConfig()	Constructor that initialize NULL the SyncSourceConfig pointer
WindowsSyncSourceConfig::WindowsSyncSourceConfig(SyncSourceConfig* sc)	Constructor that initialize the internal SyncSourceConfig to the passed one
void setCommonConfig(SyncSourceConfig* sc);	Set the internal SyncSourceConfig* variable to the DMT SyncSourceConfig* variable
SyncSourceConfig* getCommonConfig();	Set the SyncSourceConfig* variable
get/setName()	Get/Set value from SyncSourceConfig
get/setURI()	Get/Set value from SyncSourceConfig
get/setSyncModes()	Get/Set value from SyncSourceConfig
get/setType()	Get/Set value from SyncSourceConfig
get/setSync()	Get/Set value from SyncSourceConfig
get/setEncoding()	Get/Set value from SyncSourceConfig
get/setVersion()	Get/Set value from SyncSourceConfig
get/setSupportedTypes()	Get/Set value from SyncSourceConfig
get/setLast()	Get/Set value from SyncSourceConfig
get/setEncryption()	Get/Set value from SyncSourceConfig

get/setUseSif()	Get/Set value from the class
get/setSyncDir()	Get/Set value from the class
get/setSchedule()	Get/Set value from the class

The mail data source is a bit more complex than the other sources because it contains several other properties. The API C++ provides already the configuration for that data source and it is useful for the plug-in to use this one.

Therefore the ClientSettings class contains an array of WindowsSyncSourceConfigs that contain a reference to the SyncSourceConfig of the DMTClientConfig adding the custom properties of the plug-in. It also contains a MailSyncSourceConfig that is used for the mail data source. Then there are also methods of the class that are used to get other custom properties of the plug-in (*SyncclientPIM custom properties).*

- **ClientSettings**: class derived from the DMTClientConfig C++ API base class. ClientSettings reads the setting parameters that are common for all data sources and for the syncml protocol through the base class methods (see [7] for more info). It extends the basic properties with the custom needed to store particular behavior settings. The most important methods and members are:

Name	Description
ClientSettings(const char* application_uri)	Private constructor that build the configuration object starting from the *application_uri* context. The ClientSettings can only be used through a static instance of the class. In the plug-in the context is Funambol/SyncclientPIM
~ClientSettings()	Destructor
ClientSettings* ClientSettings::getInstance()	Return the instance of the ClientSettings. It creates a new one if it is the first time. It populate with all the parameters in the registry.
MailSyncSourceConfig* mailssconfig;	Pointer to the object repreenting the mail data source configuration.
WindowsSyncSourceConfig* winSourceConfigs	Pointer to an array of WindowsSyncSourceConfig containing the default properties plus the customs.
BOOL readConfig();	Read the config parameters of the SyncclientPIM level
BOOL saveConfig();	Save all the custom properties in SyncclientPIM level and also some particular properties in all the tree. They are: username, password, url, device_id, enableCompression, log level, devinf hash.
BOOL save();	Save all the settings parameters
BOOL read();	Read all the settings parameters
const char* getConfigSourcesParameter(const char* sourceName, const char* parameter);	Get a value of a parameter of a specific sync source
BOOL setConfigSourcesParameter(const char* sourceName, const char* parameter, const char* value)	Get a value of a parameter of a specific sync source
BOOL saveSyncSourceConfig(const char* name);	Save the configuration of the source plus the custom source parameters
void setDirty(int flag);	Set a dirty flag to specify which parameters are modified
BOOL saveDirty();	Save config parameters based on the dirty flag.
__declspec(dllexport) ClientSettings* getRegConfig();	Return the pointer at the static instance of the class
get/setIp();	Get/Set custom property in SyncclientPIM level
get/setPush();	Get/Set custom property in SyncclientPIM level
get/setPushPort()	Get/Set custom property in SyncclientPIM level
get/setSms()	Get/Set custom property in SyncclientPIM level
get/setPolling()	Get/Set custom property in SyncclientPIM level
get/setSvrNotified()	Get/Set custom property in SyncclientPIM level

get/setPath()	Get/Set custom property in SyncclientPIM level
get/setCradleNotification()	Get/Set custom property in SyncclientPIM level
get/setIsPortal()	Get/Set custom property in SyncclientPIM level

- **SettingsFunction**: collection of functions used to read and write single parameter in the configuration tree. It contains also functions to retrieve the IMEI (for the smartphone) and for the deviceID (pocket pc).

Name	***Description***
wstring getIMEI()	Return the IMEI of the smartphone
wstring GetSerialNumberFromKernelIoControl()	Return the device id of the pocket pc
DWORD getClientConfigurationInternal (wchar_t* sourceName, wchar_t* propertyName, wchar_t* propertyValue, BOOL isFrom)	Read a single *value* of a property *name* of the particular *sourceName.* IsFrom must be null.
DWORD WINAPI setClientConfiguration(wchar_t* sourceName, wchar_t* propertyName, wchar_t* propertyValue, wchar_t* errorMsg)	Set a single *value* of a property *name* of the particular *sourceName*. ErrorMsg must be null.

4.2. Startsync.exe

It is the main program that is started by all the agent that can start a sync. Every sync initializer fires the startsync.exe program using the *startcmd* method provided by the C++ API. The initializer put its own different command line argument to discriminate which action must be performed.
This program, through these opportune parameters, is used also to add and remove services (SMS push) or schedule a sync. When the sytartsync is called, it creates a global semaphore, named mutex ([9]), in the following way:

```
CreateMutex(NULL, TRUE, TEXT("FunSyncInProgress") );
```

This avoids that multiple syncs can overlap themselves guaranteeing no data corruption.

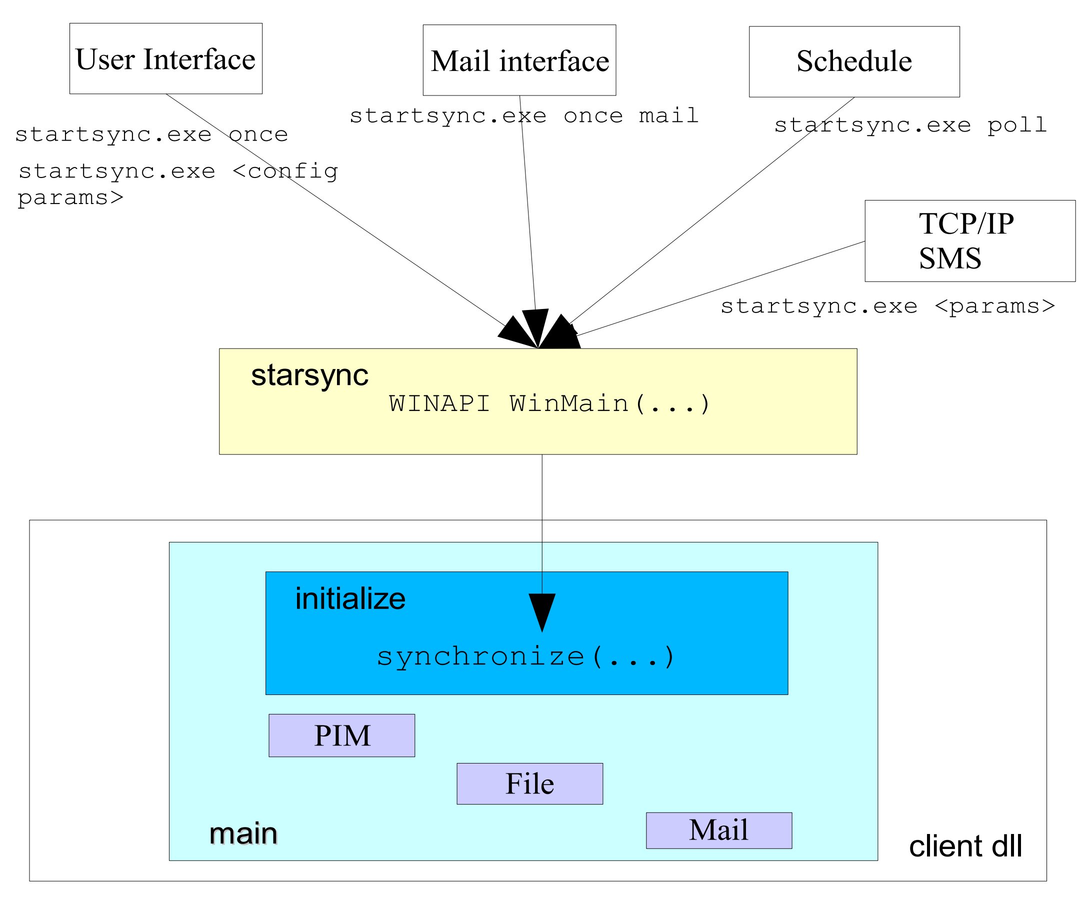


Figure 17: Startsync module and some of its caller

The first action of the WinMain function is to parse the command line to perform the right action. If the action is to start a sync, the function creates the Funambol notification icon on the tray (a man walking on the rope) and writes its own process id in a proper registry key. This will be useful to stop the sync process if the user wants to do it for every reason. Other actions permitted are to register / de-register the client to listen for the SMS push or to register / de-register the client to listen for the Over The Air configuration. The latter is to be completed.

The startsync program is usually called as follow:

```
startsync.exe <parameter <sourcename1,sourcename2> >
```

The possible command line are described below

Parameter	***Description***
poll	Activate the scheduler for the automatic sync. The time for every sync is written in a proper registry
once <sourcename1,sourcename2...>	If only “once” is specified as command line the sync will start for every data source the user have selected in the configuration panel. Otherwise only the specified sources will be synced
addresschange <sourcename1, sourcename2...>	Communicate to the server that the ip of the device is changed. Usually the command line is “addresschange mail” (TBR does mail really need?. Only addresschange should be enough)
/notify	Used when the “sync when cradled” option is checked. This parameter permits to start the sync when the device is cradled on the pc
APP_RUN_AT_TIME	Used by the system function *CeRunAppAtTime* when the schedule option is activated.

inclusive mail,/id1,/id2,/id3	Used by the syncmltransport functions (see 4.4) to complete the downloading of a partial email. Enable the mail INCLUSIVE filter
register	Register the startsync application itself as a listener for the SMS wap push server messages
deregister	Deregister the startsync application itself to listen for the SMS wap push server messages
wap_push	Used by the operating system when a SMS wap push message comes on the device. The startsync parses the sms and the starts the sync.
registerOTA	Register the startsync application itself as a listener for the OTA wap push server messages. No yet completed
deregisterOTA	Deregister the startsync application itself to listen for the OTA wap push server messages. Not yet completed
OTA_config	Used by the operating system when a SMS OTA wap push message comes on the device. It configures the main settings automatically. Not yet completed
refresh-from-server	Used by the UI functions when a recover from server is invoked
refresh-from-client	Used by the UI functions when a recover from client is invoked
available	Not used
removeNotif	Remove the notification icon

4.3. Notification listener (ntlstnr)

The Windows Mobile plug-in is able to start a sync process after a notification about some server items modifications, PIM or mail. If the user chooses this option, the plug-in says that it is reachable, informing the server about its ip address and the port on which it is listening to the incoming server messages. In order to do these actions the plug-in has a dll library responsible to communicate the current ip address (and its changes) to the server, and to wait for incoming requests.

The component that waits for TCP/IP requests is called Notification Listener, and it's a simple internet daemon waiting on a TCP port. The default port number, registered at the IANA ([10]) for the Funambol Mobile Push service, is 4745.
On Windows Mobile, is it possible to implement an internet daemon without having a process or thread running using the *service.exe ([11])*, that behaves somewhat like the inetd service on Unix machines: waits for requests on different ports and, if a request arrives, calls the service registered for that port.

The interface between service.exe and the user services is based on DLL registration: for the Funambol plugin, this interface is implemented by the notlstnr.dll module.
The DLL to be registered on service.exe must have implemented a number of functions that will be called by the OS when certain events occur (connection open/closed by the client, client reads, client writes, initialization, I/O event from the network).
Each service registered is also identified by a three-char prefix, which can be used to identify the connection as a peripheral (like PRN:). For the Funambol plugin, the prefix is S4N (Sync4J Notification). All the initerface method are prefixed by the service prefix.
Because the client-side interface is not used by the notlstnr, not all the interfaces are implemented:

Name	***Description***
DllMail	Dll entry point
S4N_Open	Not implemented
S4N_Close	Not implemented
S4N_Init	Start the thread listener
S4N_Deinit	Stop the thread listener
S4N_IOControl	Method that notify the listener with OS events. The one the plug-in handles are IOCTL_SERVICE_NOTIFY_ADDR_CHANGE IOCTL_SERVICE_REFRESH IOCTL_SERVICE_STARTED IOCTL_SERVICE_CONNECTION
S4N_Read	Not implemented
S4N_Seek	Not implemented
S4N_Write	Not implemented

To notify the ip address change the focus is on the IOCTL_SERVICE_NOTIFY_ADDR_CHANGE (called by service.exe each time the IP address changes for some reason), the IOCTL_SERVICE_REFRESH (the service S4N has been restarted) and the OCTL_SERVICE_STARTED (called by service.exe at the end of the startup procedure) event. When one of them is fired by the OS, the plug-in calls a procedure to start a particular sync process that aims to notify the server about the new ip where the device is reachable and the port on which there is a service listening.
These methods are executed in the service.exe thread, so they must process quickly the request and return the control. If a longer process is needed, it must be done in a separate thread.

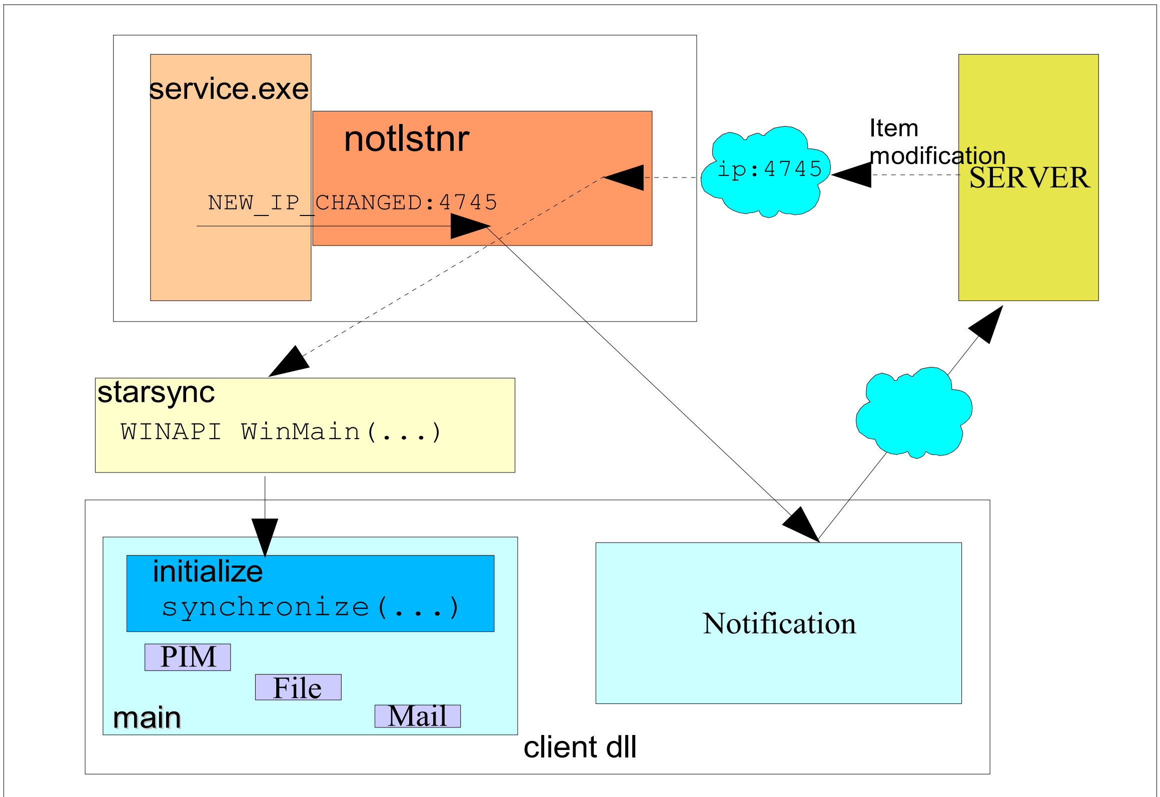


Figure 18: Notification listener interactions

When the library is registered and it has communicated its ip address and the port where it is listening, the plug-in is ready to wait for the incoming messages. If the server detects a change to be notified, like a new email is received or there is a modification on some PIM items, the server asks the client to start a synchronization sending a particular message via TCP/IP to the ip and port that the client previously sent. At this event, the IOCTL_SERVICE_CONNECTION, the library reacts parsing the message and starting the sync.

The dll library is composed by several classes and the most important ones are described below.

- **notlstnr**: the main class that implements the methods of the service.exe interface. See before about the interface method description. Follows the handler functions of the events

Name	Description
DWORD addressChangeHandler(void)	Start a particular sync saying to the server the new client IP address and the listening port. It avoid to communicate local unreachable ip (192.168.55.101 and 169.254.2.1).
static DWORD acceptConnection(SOCKET sock)	Accept the server alert, calls the right methods to parse the message and if the request is fine to start the sync process

- **s4jproxy**: singleton class containing the methods to parse the message and to start the synchronization process

Name	Description
RetCode S4JProxy::parsePkg0(const char *msg, int msglen)	Parse the server message (the package 0 in syncML language)
int S4JProxy::sync(bool debug)	Call the right function to start the sync process

- **worker**: class that is invoked by the notification library when a server request comes. It create the thread that is responsible to parse the message and to start the sync. In order to do this, it uses the s4jproxy instance class.

Name	Description
DWORD startWorker(SOCKET s)	Start the whole procedure when an incoming message arrives (start the NotifWorker thread)
DWORD stopWorker()	Dispose everythings when the service.exe is deregister and removed
const char *processPacket(const char *pkt, int pktlen)	Process the package message through the s4jproxy class
extern "C" DWORD WINAPI NotifWorker(LPVOID lpv)	The main thread for the notification server message

4.4. Syncmltransport dll

The SyncMLTransport is a dll library specific for the mail data source. It is a custom implementation of a mail transport layer as the POP3 or IMAP4 component. It permits the creation of the custom Funambol mail account and it gives the opportunity to invoke the startsync.exe process through the usual mail interface. The dll library implements several methods that are needed to construct an Inbox transport through which implement the custom protocol. The transport layer is built starting from an example shipped with the Windows Mobile 5.0 SDK ([12]).

The *synchronize()* method of the **SyncMLTransport** is the most important and it is invoked when the user presses the usual send/receive button. The action is detected and it is possible to start the usual synchronization process calling the *startsync.exe* program with "*once mail*" command line.

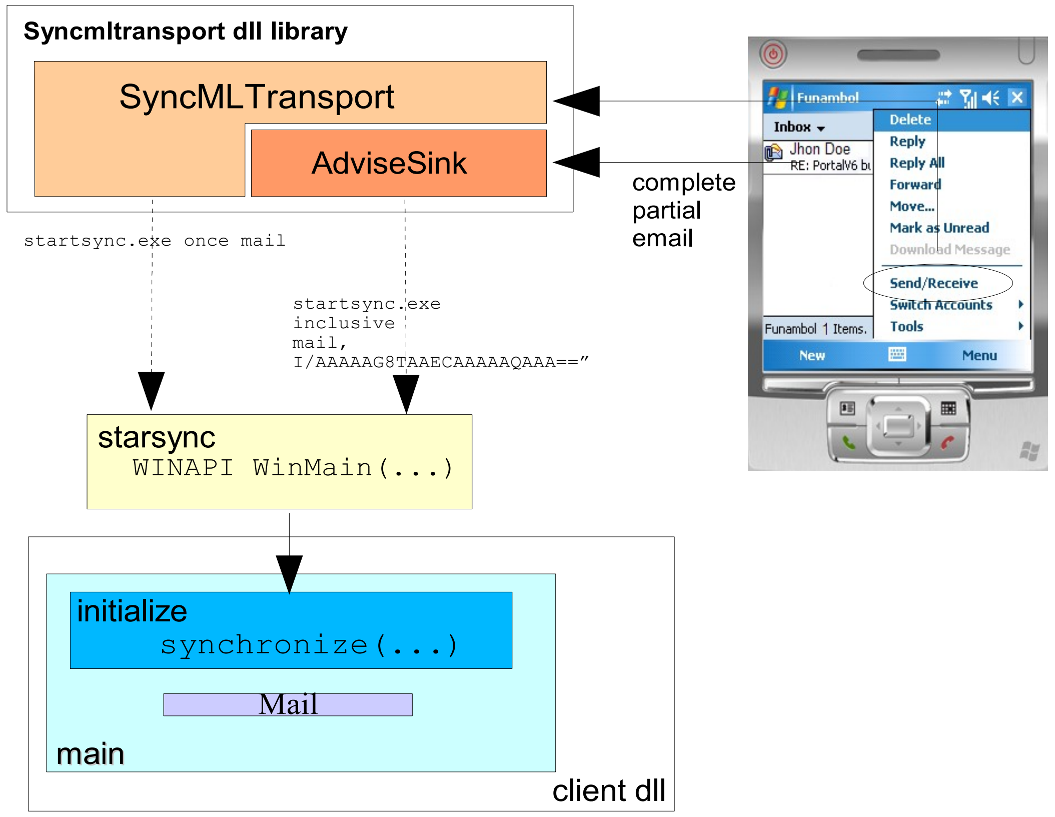


Figure 19: SyncML Transport dll library

Through the plug-in it is possible to choose some mail filter as the age of the mail, the max size and if to download the attachments too. In this way, the synchronization is easier and faster. After the user have downloaded the partial mail, he can decide to get the full email. To aim this action, in the body of the mail there is a link saying “click here to get the rest of the mail”. The click action is intercepted by another listener that is able to start to download the full email. This listener, the **AdviseSink**, creates a proper command line for the startsync.exe program specifying an inclusive filter with only the id of the mail to be completed.

- **AdviseSink:** class listener used to download the remaining part of the partial downloaded email

Name	Description
STDMETHODIMP_(ULONG) OnNotify (ULONG cNotify, LPNOTIFICATION lpNotifications)	Method called when the "click to download the rest of the message" is pressed. It runs the startsync program with command line "inclusive mail,I/AAAAAG8TAAECAAAAAQAAA==”

In order to install the transport dll library, the plug-in installation procedure writes particular registries to say to the device mail application manager that there is a new transport available for the mail account. These registry are the follows:

Name	Description
HKEY_LOCAL_MACHINE\Software\Microsoft\Inbox\Svc\Funambol\name	The name of the new transport type layer: Funambol
HKEY_LOCAL_MACHINE\Software\Microsoft\Inbox\Svc\Funambol\port	The port: 0. It is unused
HKEY_LOCAL_MACHINE\Software\Microsoft\Inbox\Svc\Funambol\DLL	The implementation of the layer: syncmltransport.dll

4.5. Handleservice

This module is a program that is invoked by several methods to start or stop some services or to create or delete the Funambol email account during the install or uninstall process. The functions are called starting the handleservice program with an appropriate command line:

```
handleservice.exe <parameter>
```

The possible parameters are listed below.

Name	Description
register	It calls the registerService methods (see 4.1.3) to register the notlstnr library
deregister	It calls the deregisterService methods (see 4.1.3) to deregister the notlstnr library
createAccount	It creates the Funambol email account. It is called by the installation procedure
deleteAccount	It delete the Funambol email account. It is called by the uninstall procedure
registerByLink	The “register” action creates a file.lnk and put in the /Windows/Startup directory that contains programs that are executed on system startup. So if the device is restarted, automatically the file.lnk calls handleservice.exe registerByLink and can restart the notification listener.
APP_RUN_AT_TIME	The “register” action sets a scheduled process (every 10 minutes) that checks the device connection and tries to resume if it is missing.
registerCradle	Set a listener on the “on cradle” event. When it is got the sync starts. NOTE. Available for Windows Mobile 5.0 only
deregisterCradle	Remove the listener on the “on cradle” event. Available for Windows Mobile 5.0 only

4.6. Syncmail

Component that needs only to start the sync of the mail. It is used by the TCP/IP notification to permit the device can play default notification mail sound when a new email comes on the device. Currently it is used only on the pocket pc plug-in version but also the smartphone needs this behavior.
The need to create a separate program to handle this little issue is due to a different compilation options between the native “cemapi” library and the plug-in one. So through this program it is possible to separate the different modules. For more information see [13] and [14].

This component only contains the main function that start with the command line

```
syncmail.exe once mail
```

This runs the MailSyncMessages mail API that emulate the Send/Receive press button and allow to play the sound when needed.

5. Installation

This section tries to describe the procedure to build and install the Windows Mobile plug-in. It shows the main modules needed for this purpose

5.1. Funsetup dll

It is the dll library hooked at the install/upgrade/uninstall procedure and is loaded first of all. The Windows Mobile offers the possibility to have a library that can execute some actions before the installation procedure starts. Therefore it is possible make some custom check o preliminary operations. The same library is executed also before the uninstall procedure starts. At the same way it is possible to do things, like to close the running UI, before starting the uninstall procedure. To learn more about dll see [15] and [16].
To have a similar library is needed to create a dll module and implement several particular methods. Then, in the cab installation package, it is necessary to load the library specifying its name (funsetup.dll).

The methods to implement are:

Name	Description
Install_Init	Used to specify which actions are to do before extracting the files of the plug-in
Install_Exit	Used to specify which actions are to do after the extraction of the files of the plug-in
Uninstall_Init	Used to specify which actions are to do before removing the plug-in
Uninstall_Exit	Used to specify which actions are to do after the removal of the plug-in

5.1.1. Installation process

In the installation process, the methods of the library are implemented in the following way:

Name	Description
Install_Init	Creates all the registries default settings Create the registries used to register the syncmltransport library (see 4.4)
Install_Exit	Creates the Funambol email account Creates the "briefcase" folder, the default one used to sync files.

5.1.2. Upgrade process

The upgrade process permits the keep the settings from a previous installation of the plug-in both 3.0 and 6.0.x versions. Windows Mobile 5.0 permits to figure out if a previous version of a program is installed, even if it has the same name. This is used for upgrading from 6.0.x version but not from 3.0, due to a different convention name. Moreover, in the 3.0 the structure of the settings tree was different and an extra effort to map old settings to new ones has to be done in the setup library.
There was also a 3.1.x version only for the smartphone device that already has the settings structure as the 6.0.

Name	*Description*
Install_Init	If previous version is already 6.0: - backup of the registries settings - backup of cache files of the sync If previous version is 3.0: - backup of the registries settings with mapping from 3.0 structure to new one - remove all file of the 3.0 plug-in (old visual basic program, old images...) - remove the 3.0 entries from the list of uninstall programs
Install_Exit	Restore the settings from backup Restore the cache files from backup Restore the notification settings if previous set

5.1.3. Uninstall process

The uninstall process remove completely the plug-in from the device, taking care to remove all the settings, the files related to the plug-in and the Funambol custom email account. The methods of the dll are the Uninstall_ prefixed and the actions performed are as follow:

Name	*Description*
Uninstall_Init	Delete the Funambol email account Close opened process and notification listener
Uninstall_Exit	Delete all registry settings Delete all unused files (logs, cache...)

5.2. build.xml

It is the *ant* ([17]) script file used to build all the source code files and to package the executables in a .cab file, to download and install directly on the device, or in a .exe file, to install from a laptop.
The plug-in installer has two different directory for pocket pc and smartphone version. All the instructions inside have only slightly differences but it is necessary at the moment keep them separated. Currently the plug-in is released for windows mobile 5.0 but there are also modules to create the version for 2003 OS, both for pocket pc and smartphone.
The main difference between the two builds is the smartphone one needs to be signed with a trusted certificate to be successfully installed on the device. In order to have the signed version it is necessary to follow a particular procedure with Verisign certification authority that won't be described here.

The modules used to build the pocket pc version are:

- *release.properties*: contains informations about the build version. It is necessary to increment the release.major, release.minor and build.number parameters to have the corresponding release.
- *pocketpc.properties*: contains the variable used by the ant build.xml script to build the plug-in
- *funambol-pocketpc-nsi-script.nsi:* script file used by *nsis* installer program [18] to create the executable to allow to install the plug-in from a laptop
- *setup-ppc-wm5.inf*: contains all the directive to create the .cab file installer.
- *build.xml*: ant script that leads all the actions permitted
- *ppcwm5.xml*: and script that leads all the actions permitted for pocket pc WM 5. It is included in the build.xml

For information about how to build the plug-in and what is required to have installed see [19].
To create the pocket release it is enough to go in the *install/pocketpc/build* directory of the plug-in source structure, and type ant (you must have ant installed).

The several possibility for the windows mobile 5.0 release are

Name	***Description***
forge-src	Create a zip file containing all the sources files needed to build the plug-in, C++ API included
clean-release-wm5	Clean all the directories of the previous built modules in order to start to build from a clean scenario
compile-no-checkout-release-wm5	Compile all the source code
forge-no-checkout-release-wm5	Move the compiled modules (dll library and executable) in a opportune folder. It also complete the creation of the .inf file necessary to create the cab file
create-lib-wm5	Group all in the cab file
create-installer-wm5	Create the executable responsible to install the plug-in via laptop (ActiveSync connection)
ppcwm5-release	Create the pocket pc plug-in from scratch. Actually it calls all the previous task (compile, forge, create lib, create installer)
ppcwm5-release-portal	Create the pocket pc plug-in portal version from scratch. It contains some limitation compared to usual version

Basically the actions to follow to build the smartphone are the same. The only difference is that it is not possible to create the release because of the code signing issue.

6. Appendices

6.1. Appendix A - References

[1] WM User Guide–http://download.forge.objectweb.org/sync4j/WinMobile_Plugin_UG.pdf
[2] CeRunAppAtTime - http://msdn2.microsoft.com/en-us/library/ms913957.aspx
[3] SyncML Data Synchronization Protocol, version 1.2, Open Mobile Alliance - http://www.openmobilealliance.org/release_program/ds_v112.html
[4] SyncML Server Alerted Notification, version 1.2, Open Mobile Alliance - http://www.openmobilealliance.org/release_program/ds_v112.html
[5] Server Alerted Synchronization Design Document
[6] POOM object model - http://msdn.gmicrosoft.com/library/default.asp?url=/library/en-us/wceappservices5/html/wce50oripocketoutlookobjectmodelpoomapi.asp
[7] Funambol C++ Design Document - http://cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/sync4j/3x/client-api/native/design/Funambol%203.0%20Client%20API%20C%2B%2B%20Design%20Document.odt
[8] Funambol ds server Developers Guide - http://cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/sync4j/3x/docs/ds-server/manual/funambol_ds_server_developer_guide.pdf
[9] Mutex - http://msdn2.microsoft.com/en-us/library/ms682411.aspx
[10] IANA - http://www.iana.org/assignments/port-numbers
[11] Service.exe - http://msdn2.microsoft.com/en-us/library/aa446909.aspx
[12] TransportDemo - http://msdn2.microsoft.com/en-us/library/ms880664.aspx
[13] MailSyncMessages - http://msdn2.microsoft.com/en-us/library/ms894668.aspx
[14] MailSyncMessages build problem - http://windowsmobilepro.blogspot.com/2006/04/windows-mobile-50-new-mapi-functions.html
[15] Setup.dll - http://www.pocketpcdn.com/articles/setupdll.html
[16] Setup2.dll - http://msdn2.microsoft.com/en-us/library/ms838599.aspx
[17] Ant - http://ant.apache.org/
[18] nsis - http://nsis.sourceforge.net/Main_Page
[19] wiki - http://wiki.objectweb.org/sync4j/Wiki.jsp?page=HowtobuildWMPlugin

6.2. Appendix B – Calendar Exception Handling

This section analyzes how the calendar exceptions are handled by the Windows Mobile plug-in. An appointment event can be set to be recursive and in the event editor it is possible to set different type of recurrences. So, for example, it is possible to set the appointment recurs every day, or every Monday and Friday for 15 times or every first Sunday of March and July. In some cases, the user could need to modify an **occurrence** of the recurring events maybe changing the subject or the start date or still delete that occurrence.

The Pocket Outlook Object Model, at the user's modification action, reacts creating an Exception object related to the principal appointment. It is possible to get that Exception object using the methods provided by the POOM API. Briefly the logical way to get the Exception object starting from the original appointment is:

```
IAppointment->IRecurrencePattern->IExceptions->IException
```

The *IException* object contains properties to understand the modification of the current occurrence and its properties are:

Name	***Description***
IException::get_AppointmentItem	Gets the Appointment item that corresponds to this exception
IException::get_OriginalDate	Gets the date that this exception originally occurred
IException::get_Deleted	Returns TRUE if the exception was caused by a deleted instance

The Windows Mobile plug-in handles the Exceptions according with other Funambol plug-ins, like Outlook, and phone embedded clients. In order to achieve these goals, this way of work is quite different from the current handling of Windows Mobile built-in exceptions handling.

Sync flow of a calendar item with exceptions

Suppose there is a recurring appointment with subject “New try” and location “Paris” on the device that contains 2 exceptions: the first one is a deletion; the second has the location modified in “Rome”.

When the sync starts, the first action on the calendars data source is “to normalize” all the appointment exceptions. Therefore all the events are analyzed and the exceptions are treated in the follow way:

- if the exception is due to an occurrence deletion no action is performed.
- if the exception is due to a field modification of the occurrence, the plug-in creates a new calendar item with the fields of the exception, that is not-related to the based event.

According with the example scenario, the first exception is untouched and the second is modified so on the device there will be an appointment with Subject “New Try” that contains two exceptions that are deletion of two occurrences. There is also a new appointment with subject “New Try” with location “Rome”.

Then the sync proceeded and the item will be exchanged to the server. The appointment exceptions are represented in the SIF format as follow:

```
<appointment>
  <startdate>...</startdate>
   ....
  <Exceptions>
    <ExcludeDate>2007-05-07</ExcludeDate>  ;date format YYYY-MM-DD
    <ExcludeDate>2007-05-09</ExcludeDate>
    <IncludeDate/>
  </Exceptions>
</appointment>
```

The <ExcludeDate> is the tag in which there are the date of the occurrence that are deleted.
The <IncludeDate> is the tag in which there are the date of the occurrence that are added. For the Windows Mobile, the POOM architecture doesn't allow to add and exception to an existing appointment. This tag will be always empty.

Then the sync proceeded and the server sends its own modification. At the same way another appointment with recurrence exceptions can come on the device. Suppose the SIF is as follow

```
<appointment>
  <startdate>...</startdate>
   ....
  <Exceptions>
```

```
    <ExcludeDate>2007-06-04</ExcludeDate>
    <IncludeDate>2007-05-10</IncludeDate>
  </Exceptions>
</appointment>
```

The plug in processes the Exception tags as follows:

- ExcludeDate: the plug-in deletes the occurrence at that date.
- IncludeDate: the plug-in treats these as new appointments that are not-related to the principal one.