SyncML WSPBinding

Approved Version 1.2 – 21 Feb 2007

Open Mobile Alliance

OMA-TS-SyncML_WSPBinding-V1_2-20070221-A

Contents

Figures

1. Scope

The SyncML Initiative, Ltd. was a not-for-profit corporation formed by a group of companies who co-operated to produce an open specification for data synchronization and device management. Prior to SyncML, data synchronization and device management had been based on a set of different, proprietary protocols, each functioning only with a very limited number of devices, systems and data types. These non-interoperable technologies have complicated the tasks of users, manufacturers, service providers, and developers. Further, a proliferation of different, proprietary data synchronization and device management protocols has placed barriers to the extended use of mobile devices, has restricted data access and delivery and limited the mobility of the users.

SyncML Components

SyncML is a specification that contains the following main components:

- An XML-based representation protocol
- A synchronization protocol and a device management protocol
- Transport bindings for the protocol

The data representation specifies an XML DTD that allows the representation of all the information required to perform synchronization or device management, including data, metadata and commands. The synchronization and device management protocols specify how SyncML messages conforming to the DTD are exchanged in order to allow a SyncML client and server to exchange additions, deletes, updates and other status information.

There are also DTDs which define the representation of information about the device such as memory capacity, and the representation of various types of meta information such as security credentials.

Although the SyncML specification defines transport bindings that specify how to use a particular transport to exchange messages and responses, the SyncML representation, synchronization and device management protocols are transport-independent. Each SyncML package is completely self-contained, and could in principle be carried by any transport. The initial bindings specified are HTTP, WSP and OBEX, but there is no reason why SyncML could not be implemented using email or message queues, to list only two alternatives. Because SyncML messages are self-contained, multiple transports may be used without either the server or client devices having to be aware of the network topology. Thus, a short-range OBEX connection could be used for local connectivity, with the messages being passed on via HTTP to an Internet-hosted synchronization server.

To reduce the data size, a binary coding of SyncML based on the WAP Forum's WBXML is defined. Messages may also be passed in clear text if required. In this and other ways SyncML addresses the bandwidth and resource limitations imposed by mobile devices.

SyncML is both data type and data store independent. SyncML can carry any data type which can be represented as a MIME object. To promote interoperability between different implementations of SyncML, the specification includes the representation formats used for common PIM data.

This document is limited to the functionality description for the SyncML WSP bindings

2. References

2.1 Normative References

[IOPPROC]	"OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/
[PUSH]	"Push OTA Protocol Specification", WAP Forum, URL:http://www1.wapforum.org/tech/terms.asp?doc=WAP-235-PushOTA-20010425-a.pdf
[PUSHARCH]	"Push Architectural Overview", WAP Forum, URL:http://www1.wapforum.org/tech/terms.asp?doc=WAP-250-PushArchOverview-200010703-a.pdf
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt
[RFC2616]	"Hypertext Transfer Protocol – HTTP/1.1", IETF, RFC 2616, URL:http://www.ietf.org/rfc/rfc2616.txt
[SAN]	"SyncML Server Alerted Notification", Open Mobile Alliance™ , OMA-TS-SyncML_SAN-V1_2, URL:http://www.openmobilealliance.org/
[SYNCHTTP]	"SyncML HTTP Binding Specification", Open Mobile Alliance™, OMA-TS-SyncML_HTTPBinding-V1_2, URL:http://www.openmobilealliance.org/
[SYNCPRO]	"SyncML Synchronization Protocol", Open Mobile Alliance™, OMA-TS-DS_DataSyncProtocol-V1_2", URL:http:www.openmobilealliance.org/
[WBXML]	WAP Binary XML Content Format Specification, URL:http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WBXML-19990616.pdf
[WDP]	"Wireless Datagram Protocol Specification", WAP Forum, URL:http://www1.wapforum.org/tech/terms.asp?doc=WAP-259-WDP-20010614-a.pdf
[WSP]	"Wireless Session Protocol specification", URL:http://www.wapforum.org/WAP-230-WSP-20010705-a.pdf
[WTP]	"Wireless Transaction Protocol Specification", WAP Forum, URL:http://www1.wapforum.org/tech/terms.asp?doc=WAP-224-WTP-20010710-a.pdf
[XML]	"Extensible Markup Language (XML) 1.0", World Wide Web Consortium Recommendation, URL:http://www.w3.org/TR/REC-xml

2.2 Informative References

None.

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

Any reference to components of the SyncML DTD or XML snippets is specified in this `typeface`.

3.2 Definitions

Bearer Network	A bearer network is used to carry the messages of a transport-layer protocol - and ultimately also of the session layer protocols - between physical devices. During the lifetime of a session, several bearer networks might be used.
Capability	Capability refers to the session layer protocol facilities and configuration parameters that a client or server supports.
Capability Negotiation	Capability negotiation is the mechanism for agreeing on session functionality and protocol options. Session capabilities are negotiated during session establishment. Capability negotiation allows a server application to determine whether a client can support certain protocol facilities and configurations.
Client and Server	The term client and server are used in order to map WSP to well known and existing systems. A client is a device (or application) which initiates a request for a session. The server is a device that passively waits for session requests from client devices. The server can either accept the request or reject it. An implementation of the WSP protocol can include only client or server functions in order to minimize the footprint. A client or server may only support a subset of the protocol facilities, indicating this during protocol capability negotiation.
Connectionless Session Service	Connectionless session service is an unreliable session service. In this mode, only the request primitive is available to service users, and only the indication primitive is available to the service provider.
Connection-Mode Session Service	Connection-mode session service is a reliable session service. In this mode, both request and response primitives are available to service users, and both indication and confirm primitives are available to the service provider.
Content	The entity body sent with a request or response is referred to as content. It is encoded in a format and encoding defined by the entity-header fields.
Content Negotiation	Content negotiation is the mechanism the server uses to select the appropriate type and encoding of content when servicing a request. The type and encoding of content in any response can be negotiated. Content negotiation allows a server application to decide whether a client can support a certain form of content.
Entity	An entity is the information transferred as the payload of a request or response. An entity consists of meta-information in the form of entity-header fields and content in the form of an entity-body.
Header	A header contains meta-information. Specifically, a session header contains general information about a session that remains constant over the lifetime of a session; an entity-header contains meta-information about a particular request, response or entity body.
Loader	Entity that implements the HTTP protocol. The loader is the interface between the WSP layer and the user application.

3.3 Abbreviations

OMA	Open Mobile Alliance
PPG	Push Proxy Gateway [PUSH][PUSHARCH]
WSP	Wireless Session Protocol [WSP]

4. Introduction

This document describes how to use the SyncML over WSP (WAP). The document uses the primitives and methods defined in the WAP Forum WSP specification as of WAP June 2000 Conformance Release.

The document describes the use of the WSP layer.

The document assumes a scenario consisting of a sync client (e.g. a wap enabled mobile phone) and a server holding data (e.g. a web-server). Furthermore, it is explained how to initiate a sync-session from the server using the WAP Push.

WAP [WSP] defines both a connection oriented and a connection less services for data exchange. Furthermore, it defines a server originated data-push model.

Note that the WAP specification does not specify the Loader, i.e. the interfaces to the different layers in the protocol, only the messages being used for communication in the actual layers. How the Loader is specified is up to the client (or server) vendor. The Loader interface is illustrated in the following by the messages going back and forth between the SyncML user agent and the Loader.

The Session layer protocol family in the WAP architecture is called the Wireless Session Protocol, WSP. WSP provides the upper-level application layer of WAP with a consistent interface for two session services. The first is a connection-mode service that operates above a transaction layer protocol WTP, and the second is a connectionless service that operates above a secure or non-secure datagram transport service. For more information on the transaction and transport services, please refer to[WTP] *“Wireless Application Protocol: Wireless Transaction Protocol Specificatio*n” and [WDP] [WDP]“*Wireless Application Protocol: Wireless Datagram Protocol Specificatio*n”. WSP provides HTTP 1.1 functionality and incorporates new features such as long-lived sessions, a common facility for data push, capability negotiation and session suspend/resume. The protocols in the WSP family are optimised for low-bandwidth bearer networks with relatively long latency.

5. WSP mapping to SyncML

The following sections define the requirements for the binding of SyncML to WSP.

5.1 Multiple messages Per Package

The WAP protocol expects to receive a response to every request sent to the WAP gateway. If there are multiple messages in a SyncML package to be sent, the SyncML server **MUST** send a response to each message although the message is not the final one.

The next message can only be sent when the WSP layer in the WAP protocol has received a response.

Each SyncML message MUST be transferred as a SyncML MIME media type within the body of a WSP request or response. When there are multiple SyncML messages per SyncML package, each message is transferred in a separate WSP request or response; depending on whether it is a SyncML request or response.

The recipient of a SyncML package can determine if there are more SyncML messages in the package by the absence of the Final element in the body of the last received SyncML message. When the recipient receives a SyncML message with the Final element, it is the final message within that SyncML package.

5.2 MIME header type requirement

Data synchronization client implementations conforming to this specification MUST support this header with either the "application/vnd.syncml+xml" or "application/vnd.syncml+wbxml" media type values. Data synchronization server implementations conforming to this specification MUST support both "application/vnd.syncml+xml" and "application/vnd.syncml+wbxml" media type values, as requested by the SyncML data synchronization client.

Device Management client implementations conforming to this specification MUST support this header with either the "application/vnd.syncml.dm+xml" or "application/vnd.syncml.dm+wbxml" media type values. Device management server implementations conforming to this specification MUST support both "application/vnd.syncml.dm+xml" and "application/vnd.syncml.dm+wbxml" media type values, as requested by the SyncML device management client.

5.3 Connection Oriented Session

This section describes how SyncML user agent residing on a WAP client would initiate a SyncML connection oriented session, exchange data with the server, suspend and resume the session, and then finally close down the established session.

5.3.1 Session establishment, S-Connect

During a WAP session establishment, a WAP client connects to a WAP gateway. A part of this is the so-called capability negotiation, during which the server and client negotiate the features supported. Furthermore, attributes that are static throughout the sessions are exchanged (static headers).

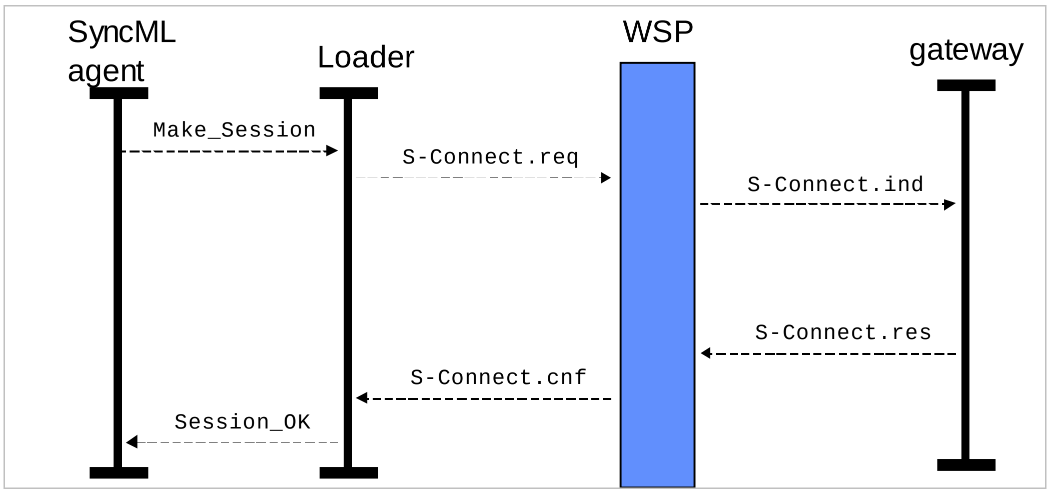

Figure 1 Session Establishment

In the example, the Loader implements an interface for the user agent to initiate a session, Make_Session. The Loader implements the HTTP protocol.

Seen from WSP, the session establishment starts by an S-Connect request to the WSP layer. The request looks as follows:

```
S-Connect.req(Server-Address , Client-Address, Client-Headers, Requested-
Capabilities)
```

In case of success, the connect confirmation returns from the WSP layer as follows:

```
S-Connect.cnf(Server-Headers , Negotiated-Capabilities)
```

5.3.2 Exchanging SyncML Data

Once a session is established, the client can start exchanging data with the server using the S-MethodInvoke and S-MethodResult primitives.

WAP maps the HTTP 1.1 methods; i.e. requests will be done using standard HTTP 1.1 methods. The header and bodies of the HTTP methods are not used by the WAP stack, and they are passed transparently.

The following example shows a simple POST request from the client.

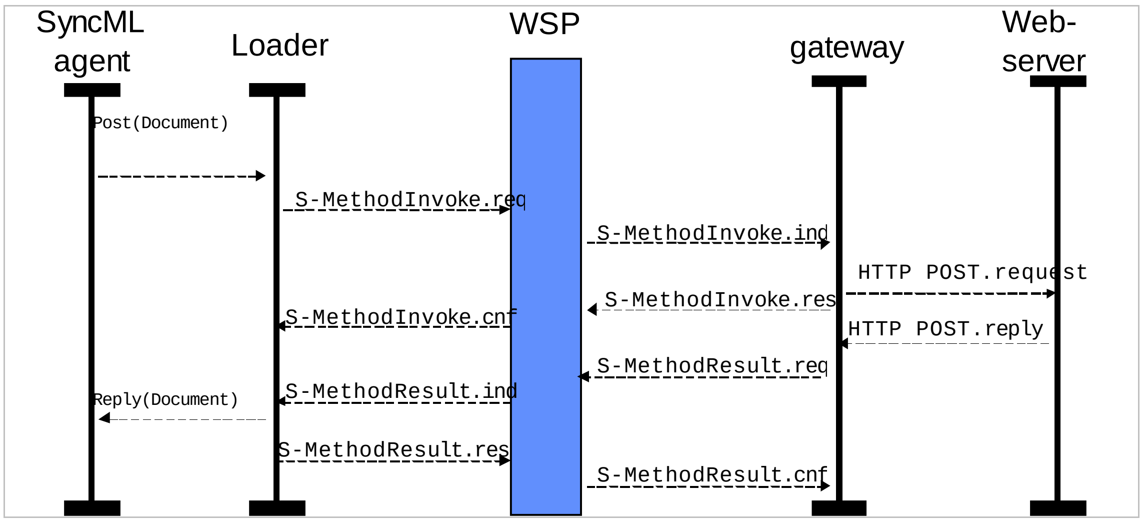

Figure 2 MethodInvoke using HTTP POST

In the implementation example depicted in Figure 2, the SyncML user agent requests a SyncML document to be posted to the server using the interface made available by the Loader. In a response, the server returns a response SyncML document back to the client.

5.3.2.1 HTTP header requirement

The HTTP header information is passed transparently over the WAP protocol. But In order to enable the Web server to decode the posted information the same header information requirements apply for sending SyncML over WSP as for sending SyncML over HTTP as described in [SYNCHTTP].

5.3.2.2 S-MethodInvoke

The syntax of the MethodInvoke is as follows:

```
S-MethodInvoke.req(ClientTransActionID, Method, RequestURI, RequestHeaders,
RequestBody)
```

The HTTP methods supported by WSP are GET, OPTIONS, HEAD, DELETE, TRACE, POST and PUT. Of all the HTTP methods supported by WSP, the SyncML functionality only requires the POST method. Once the gateway has processed the request (i.e. forwarded it to the web-server), a confirmation is sent back to the client through the WSP layer. The syntax of the S-MethodInvoke-confirmation is:

```
S-MethodInvoke.cnf(ClientTransactionID)
```

5.3.2.3 S-MethodResult

When the gateway receives the resource requested with the S-MethodInvoke primitive, it send a S-MethodResult request to the WSP layer of the client, which forwards the request to the user agent as a S-MethodResult-indication of the following format:

```
S-MethodResult.ind(ClientTransactionID, Status, ResponseHeaders, ResponseBody)
```

Once the indication is received, the client SHOULD reply to the WSP with a S-MethodResult response:

```
S-MethodResult.res(ClientTransactionID, Acknowledgement Headers)
```

5.3.3 Temporarily suspending the session, S-Suspend and S-Resume

WSP allows for the application layer to suspend a session. Suspending a session means that the sessions can no longer be used to communicate through until the session is resumed.

```
S-Suspend.req()
```

The indication coming back from WSP is of the following format:

```
S-Suspend.req(Reason)
```

5.3.4 Session close-down, S-Disconnect

The Disconnect primitive is used for terminating the active session.

5.4 Connectionless service

The connectionless service offered by WSP offers a connectionless, and potentially unreliable, data exchange service. Following example shows a POST request using the connectionless service.

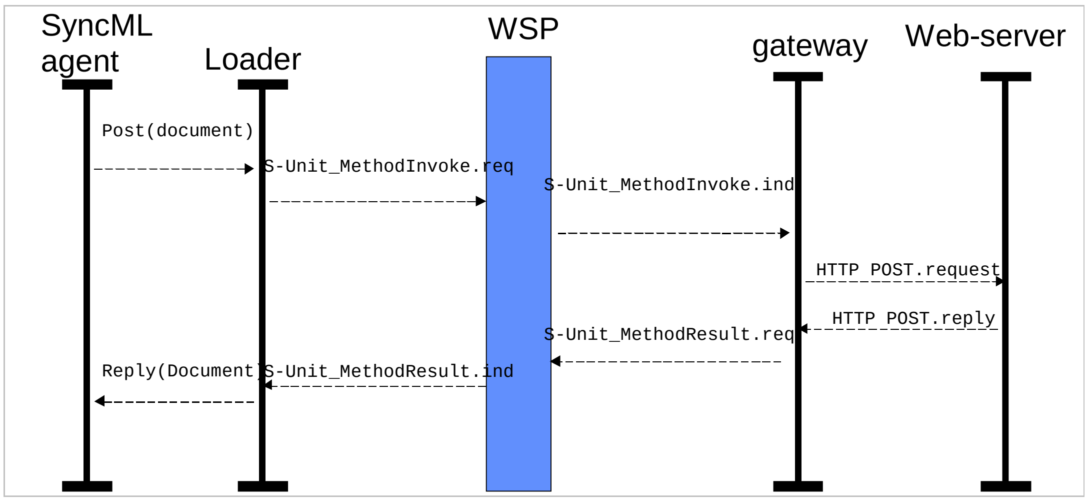

Figure 3 Connectionless Unit_MethodInvoke using HTTP POST

Only two primitives are supported by the connectionless service, MethodInvoke and Push. They both work as with the Session Oriented Service, but without the confirmation. Refer to the Session Oriented Service for details.

5.5 Pushing data from the server to the client

Pushing data from a server to a client, using Push OTA (WSP or HTTP) can be used to accomplish Server Alerted Notification as defined in the SyncML Server Alerted Notification specification [SAN]. The model is as shown below.

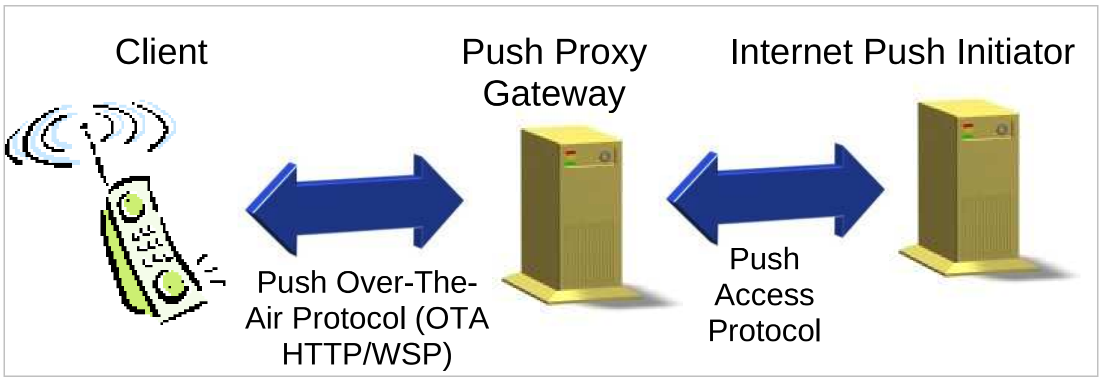

Figure 4 Push Model

The initiator of the push uses HTTP POST to send a XML document to the Push Proxy Gateway (PPG). The push message consists of two components; a control entity (containing e.g. information about when the push message expires) and the content to be pushed to the client. All WAP content types can be pushed.

When pushing SyncML data from the server to the client, the following rules MUST be adhered to:

- The Content-Type header MUST include the MIME media type, *application/vnd.syncml.notification* for Device Management usage for the Server Alerted Notification package as defined in [SAN] and *application/vnd.syncml.ds.notification*)for Data Synchronization usage. The appropriate Content-Type binary code MUST be used instead of its textual representation, that is, 0x44 for Device Managament and 0x4E for Data Synchronization.

- The X-WAP-Application-ID header MUST include the application-id x-wap-application:syncml.dm for Device Management packages or x-wap-application:push.syncml for Data Synchronization packages. The appropriate application-id binary code is used instead of the textual representation of the Application-id, that is 0x05 for Data Synchronization and 0x07 for Device Management.

The PUSH model defined two different modes; confirmed and non-confirmed. The confirmed push requires an active WSP session. If no such session is running, a session establishment request can be issued from the PPG to a special application residing in the client. This application will initiate the session, where-after the push can be accomplished.

The non-confirmed push messages does not require a session.

A special push dispatcher in the client receives the push message, and forwards it to the right application, determined by the application identifier in the push message.

Once the right application receives the push content, it might choose to pull more content from a server. This is done using the standard WSP protocols as described in this document.

As such, the client that receives the pushed content doesn't interface directly to WSP.

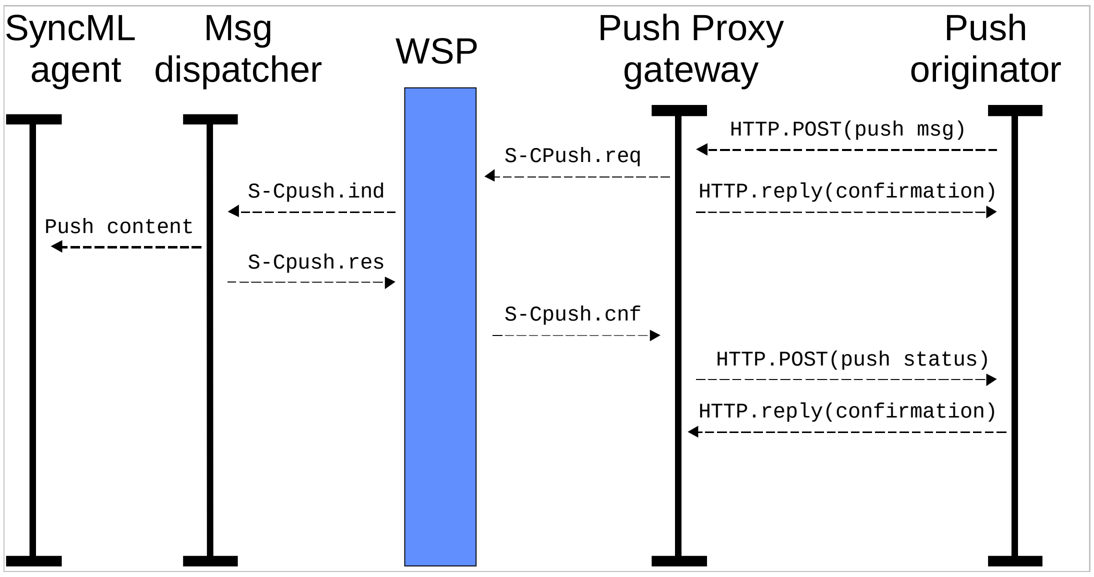

Figure 5 Push scenario

6. Examples

The following is an example WAP Push over SMS containing the Server Alerted Notification information for Device Management:

Binary value	**Meaning**	**Description**
06	User-Data-Header (UDHL) Length = 6 bytes	WDP layer (start WDP headers).
05	UDH IE identifier: Port numbers	
04	UDH port number IE length	
0B	Destination port (high)	Port number 2948
84	**Destination port (low)**	
C0	Originating port (high)	Port number chosen by sender
02	Originating port (low)	WDP layer (end WDP headers)
01	**Transaction ID / Push ID**	WSP layer (start WSP headers)
06	PDU type (push)	
03	Headerslength (content type+headers)	
C4	Content type code	MIME-Type
AF	**X-WAP-Application-ID**	
87	Id for url: x-wap-application:syncml.dm	WSP layer (end WSP headers)
	128-bit digest value	Digest
	Binary ‘0000001011’	Version ‘1.1’
	Binary ‘01’	UI-Mode ‘1’
	Binary ‘0’	Initiator ‘0’
	Binary ‘000000000000000000000000000’	Future DM use
	Binary ‘0000000000000001’	SessionID ‘1’
	Binary ‘00010010’	Server Identifier length ‘18’
63, 6F, 6D, 2E, 6D, 67, 6D, 74, 73, 72, 76, 2E, 6D, 61, 6E, 61, 67, 65	String ‘com.mgmtsrv.manage’	Server Identifier

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-SyncML-WSPBinding-V1_1_2-20030612-A	12 Jun 2003	Initial OMA release
OMA-TS-SyncML-WSPBinding-V1_2-20070221-A	21 Feb 2007	Status changed to Approved by TP: OMA-TP-2007-0005R03-INP_ERP_SyncML_Common_V1_2_for_Final_Approval

Appendix B. Static Conformance Requirements (Normative)

The following specifies the static conformance requirements for SyncML over WSP devices conforming to this specification. The notation used in this appendix is specified in [IOPPROC].

B.1 Client Features

B.1.1 Client WSP Method Requirements

Item	Function	Ref.	Status	Requirement
DSDM-WSP-C-001	Support for POST method	5.3	M	
DSDM-WSP-C-002	Support for PUSH operation	5.5	O	

B.2 Server Features

B.2.1 Server WSP Method Requirements

Item	Function	Ref.	Status	Requirement
DSDM-WSP-S-001	Support for POST method	5.3	M	
DSDM-WSP-S-002	Support for PUSH operation	5.5	O	