SyncML WSP Binding, Version 1.1

Abstract

This specification defines how to use the SyncML in a WAP environment, i.e. it specifies how to use SyncML over WSP (Wireless Session Protocol). It also describes how to initiate a sync session from a server using the push protocol specified in the WAP June 2000 conformance Release.

SyncML Initiative

The following companies are sponsors in the SyncML Initiative:

Ericsson
IBM
Lotus
Matsushita Communications Industrial Co., Ltd.
Motorola
Nokia
Openwave
Starfish Software
Symbian

Revision History

Revision	Date	Comments
V1.0 a	2000-08-14	Reformatted for v1.0 Alpha release. No technical changes.
V1.0 b	2000-11-07	Reformatted for v1.0 Beta release. No technical changes.
V1.0	2000-12-07	The draft document warning has been removed. No technical changes.
V1.0.1 a	2001-05-24	Updated from errata.
V1.0.1b	2001-05-25	Minor changes (e.g. fixing hyperlinks).
V1.0.1	2001-05-30	Minor cleanup.
V1.1	2002-02-15	Updated with 1.1 modifications

Copyright Notice

Table of Contents

1 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.

2 Formatting Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [5].

Any reference to components of the Device Information DTD or XML snipets are specified in this `type face`.

3 Terminology

The following terms are copied from [1].

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 may 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 may include only client or server functions in order to minimise 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.

4 WSP Introduction

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 [6] *"Wireless Application Protocol: Wireless Transaction Protocol Specificatio*n"and [7] *"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 an 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).

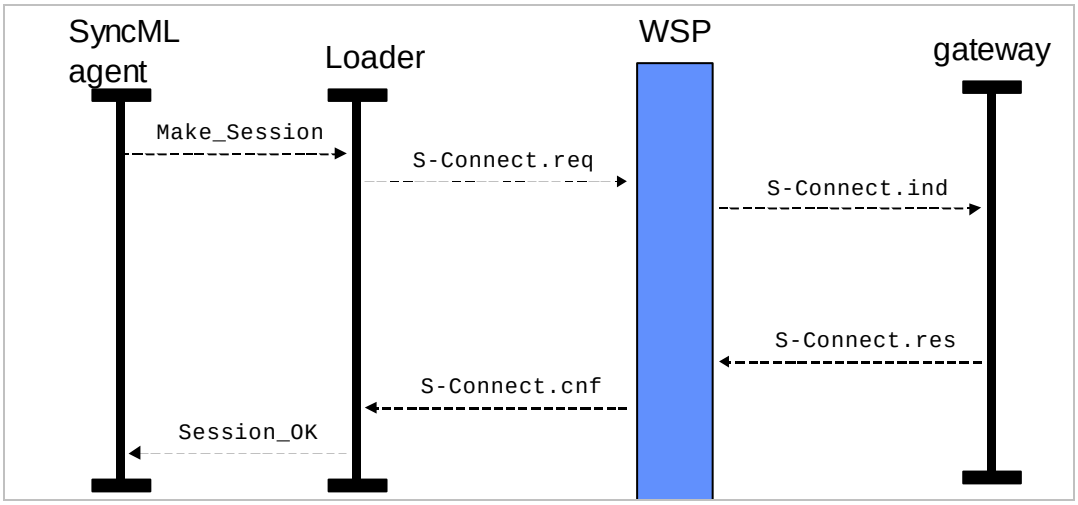

Figure 1 Session Establishment

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.

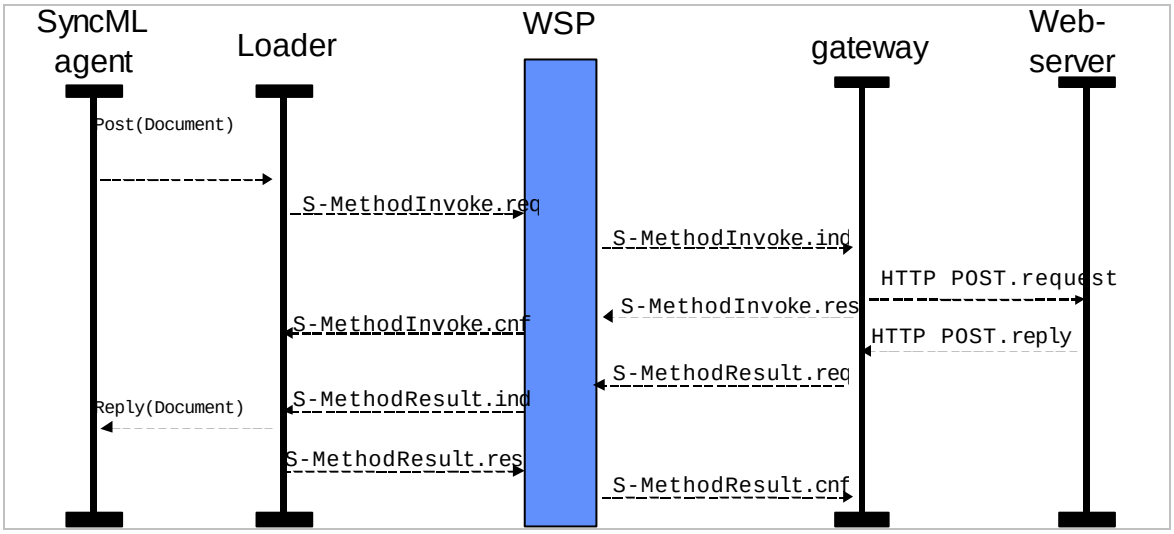

Figure 2 MethodInvoke using HTTP POST

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 transparantly 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 [9]

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.

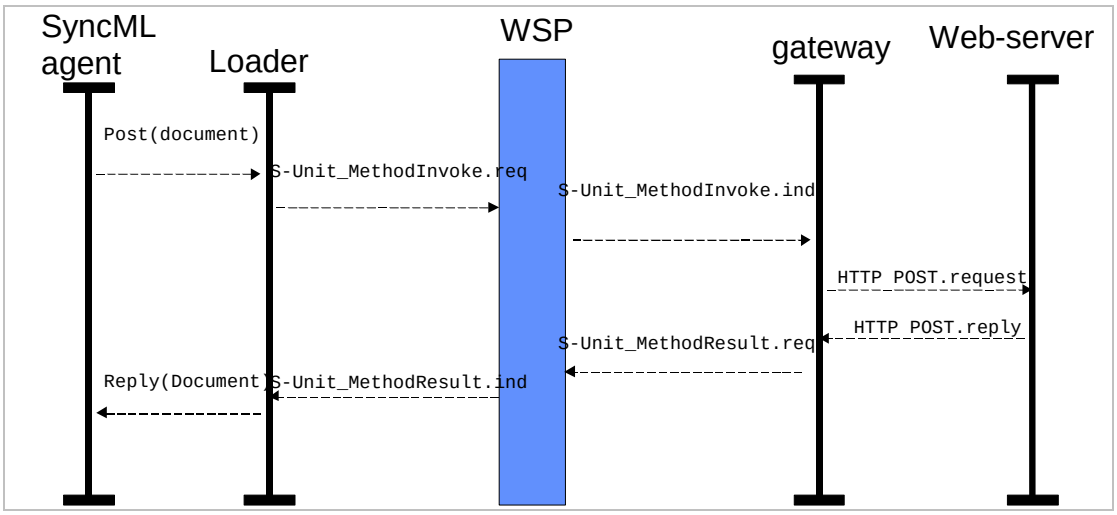

Figure 3 Connectionless Unit_MethodInvoke using HTTP POST

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 in WAP is currently only defined using HTTP. This functionality can be used to accomplish the Server Alerted Sync as defined in the SyncML Sync protocol specification [8] The model is as shown below.

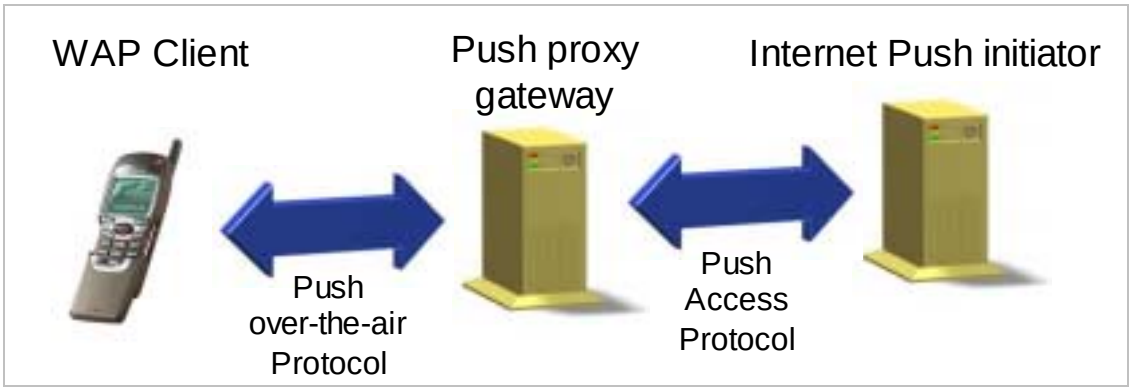

Figure 4 Push Model

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 PUSH id 0x05 MUST be used and either of the content types defined in chapter 5.2 MUST be used.

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.

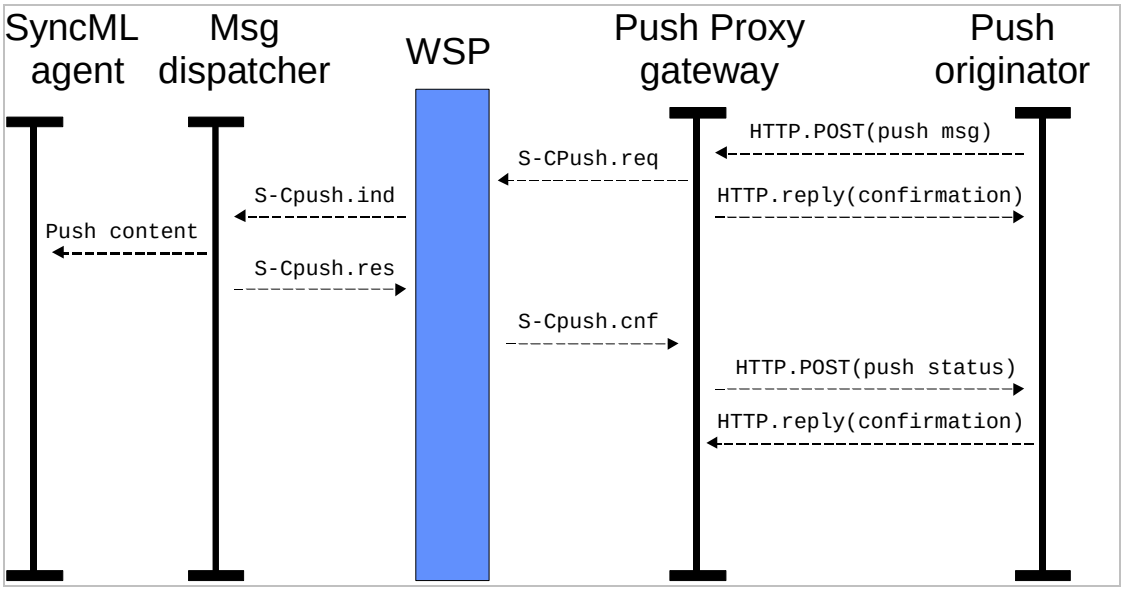

Figure 5 Push scenario

6 Static Conformance Requirements

Static conformance requirements (SCR) specify the features that are optional, mandatory and recommended within implementations conforming to this specification.

A simple table are used to specify this information.

In this table, optional features are specified by a "MAY", mandatory features are specified by a "MUST" and recommended features are specified by a "SHOULD". An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality.

The following specifies the static conformance requirements for SyncML over WSP devices conforming to this specification.

Element Type	Support of Synchronization Server		Support of Synchronization Client	
	Sending	Receiving	Sending	Receiving
POST	MUST	MUST	MUST	MUST
PUSH	MAY	MAY	- - -	MAY

7 Abbreviations

[WSP] Wireless Session Protocol.

[PPG] Push Proxy Gateway.

[WAP] Wireless Application Protocol.

[HTTP] Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, IETF.

[WBXML] WAP Binary XML Content Format Specification, WAP Forum.

[XML] Extensible Markup Language (XML) 1.0, W3C.

8 References

[1] Wireless Session Protocol specification, Approved version 15 May 2001, WAP Forum

[2] Push Architectural Overview, approved version November 8 1999, WAP Forum

[3] Push Access Protocol, approved version November 8 1999, WAP Forum

[4] HTTP 1.1 specification, rfc2068 Levels, IETF

[5] Key words for use in RFCs to Indicate Requirement Levels, IETF

[6] Wireless Transaction Protocol Specification, approved version 11 June 1999, WAP Forum

[7] Wireless Datagram Protocol Specification, approved version 05 November 1999 WAP Forum

[8] SyncML Synchronization Protocol, SyncML

[9] SyncML HTTP Binding, SyncML