SyncML HTTP Binding, version 1.1

Abstract

This document describes how to use the SyncML over HTTP. The document uses the primitives and methods defined in the HTTP specification V1.1 as defined in [1].

SyncML Initiative

The following companies are Sponsors of the SyncML Initiative:

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

Revision History

Revision	Date	Comments
V1.0	2000-12-07	The candidate version for the final release.
V1.0.1	2001-05-29	Updated copyrights, fixed media type definitions.
V1.1	2002-02-15	1.1 Release, updated legal text, updated SCRs, updated links, added MIME types.

Copyright Notice

Table of Contents

1 Introduction

This document defines the binding requirements for communicating SyncML over the Hypertext Transfer Protocol (HTTP) as defined by [1]. HTTP is an application-level protocol for distributing information between two Internet based applications. HTTP can be an ideal protocol for building wide area, distributed systems, such as the collaborative framework of the World-Wide Web (Web) environment. HTTP is expected to be the primary wireline protocol used by SyncML clients and servers to communicate with each other. In addition, as wireless networks increase in bandwidth (e.g., deployment of GPRS networks) and are adapted to support the IP protocol, HTTP will become the preferred application protocol for connecting wireless SyncML clients and servers.

The HTTP protocol defines a request/response form of communication between two network computers supporting HTTP applications. Normally, the originator of the request is called the HTTP client. And normally, the recipient of the request is called the HTTP server.

There are emerging Internet standards for notification or "push" technologies that will allow HTTP servers to also originate HTTP requests. However, this technology is not currently included in this version of the specification.

HTTP is an ideal protocol for connecting a SyncML client to a SyncML server; and vice a versa. SyncML is intended to be registered as a MIME media type. This makes it adapted for transfer over HTTP. HTTP is widely supported across the Internet. There are numerous examples of common implementations of both client and server HTTP applications. HTTP servers can be adapted to support new HTTP-based application in a straightforward manner. SyncML messages can be transferred across the Internet and pass through "firewalls", at the perimeter of individual intranets with relative ease.

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 [3].

Any reference to components of the Device Information DTD or XML [5] snippets are specified in this `type face`.

3 Terminology

See [6] and [7] for definitions of SyncML terms used within this specification. The following terms are copied from [1].

Cache

> A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel.

Cacheable

A response is cacheable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cacheability of HTTP responses are defined in section 13 of [1]. Even if a resource is cacheable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

Connection

A transport layer virtual circuit established between two programs for the purpose of communication.

Content Negotiation

The mechanism for selecting the appropriate representation when servicing a request, as described in section 12 of [1]. The representation of entities in any response can be negotiated (including error responses).

Entity

The information transferred as the payload of a HTTP request or HTTP response. An entity consists of meta-information in the form of entity-header fields and content in the form of an entity-body, as described in section 7 of [1].

HTTP Client

A program that establishes connections for the purpose of sending HTTP requests.

NOTE: In this document, when the term "client" appears alone, it refers to a HTTP client, not a SyncML client.

HTTP Gateway

A HTTP server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

HTTP Message

The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax defined in section 4 of [1] and transmitted via the connection.

HTTP Proxy

An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy MUST implement both the client and server requirements of this specification. A "transparent proxy" is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification. A "non-transparent proxy" is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering. Except where either transparent or non-transparent behavior is explicitly stated, the HTTP proxy requirements apply to both types of proxies.

HTTP Request

An HTTP request message, as defined in section 5 of [1].

HTTP Response

An HTTP response message, as defined in section 6 of [1].

HTTP Server

An application program that accepts connections in order to service HTTP requests by sending back responses. Any given program may be capable of being both an HTTP client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. Likewise, any server may act as an HTTP origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

NOTE: In this document, when the term "server" appears alone, it refers to a HTTP server, not a SyncML server.

Inbound/Outbound

Inbound and outbound refer to the request and response paths for messages: "inbound" means "traveling toward the origin server", and "outbound" means "traveling toward the user agent"

Representation

An entity included with a response that is subject to content negotiation, as described in section 12 of [1]. There may exist multiple representations associated with a particular response status.

Resource

A network data object or service that can be identified by a URI, as defined in section 3.2 of [1]. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

User Agent

The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.

Origin Server

The HTTP server on which a given resource resides or is to be created.

Tunnel

An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

Variant

A resource may have one, or more than one, representation(s) associated with it at any given instant. Each of these representations is termed a "varriant". Use of the term "variant" does not necessarily imply that the resource is subject to content negotiation.

Upstream/Downstream

Upstream and downstream describe the flow of a message: all messages flow from upstream to downstream.

4 HTTP Introduction

HTTP communication usually takes place over a TCP/IP connection. The default TCP port for HTTP is port 80, but other unregistered ports can also be used. HTTP can also be implemented on top of any other reliable transport service.

HTTP is a request/response protocol. An HTTP client sends a request message to the HTTP server. The request message consists of start line containing a request method, target URI, and protocol version; followed by a MIME-like request header lines containing meta-information about the request; followed by a blank line; followed by a possible MIME entity body content. The server responds with a status line, including the message's protocol version and a response code; followed by MIME-like response header lines containing server information and entity meta-information; followed by a blank line; followed by a optional MIME entity body content.

Further, detailed information on HTTP can be found in [1].

5 SyncML Binding to HTTP

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

5.1 TCP Transport Service

HTTP communication usually takes place over a TCP connection. This binding does not require, but does assume such a connection. If the HTTP communication takes place over another transport service, then requirements similar to those defined here for TCP need to be followed. The following sections describe requirements for connection, disconnection and abort of the TCP connection.

5.1.1 Connection

Prior to an HTTP client connecting to an HTTP server, the SyncML client makes a TCP connection using the TCP open operation between the client machine and the server machine. The client can use the Internet address resolution for the HTTP connection URL. The default port is 80. However, another port can be used. The choice of the appropriate port can be a requirement of the provisioning of the HTTP client to the HTTP server. After creating a successful TCP connection between the client and server machines, the HTTP connection between the client and server machines can be established by the SyncML client. It can not be assumed that the TCP connection is kept open between each HTTP request and response sequence (i.e., HTTP version 1.1 or higher protocol is being used). Even though HTTP version 1.1 presumes persistent connections, there may have occurred an anomoly such as a timeout condition or "denial of service" counter-measures on the origin server. Hence, the SyncML client SHOULD be prepared for reconnection between HTTP requests.

5.1.2 Connection Options

The default port is 80. However, the provisioning of the HTTP client to the HTTP server can require use of another port.

Persistent connections are supported by this specification and presumed by HTTP version 1.1, but are not required by implementations conforming to this specification.

5.1.3 Disconnection

The SyncML client is responsible for terminating the connection with a TCP close operation, when the connection is no longer needed.

If a persistent connection exists between HTTP requests, the HTTP connection is closed by the HTTP client after receipt of the HTTP response corresponding to the last SyncML request in the synchronization session (i.e., the SyncML request in the last SyncML package containing a Final element type specified at the end of the SyncBody).

5.1.4 Abort

Sometimes abnormal conditions arise which requires an application to break the TCP connection. In these cases, the TCP reset operation is initiated to terminate the TCP connection.

5.1.5 Timeouts

In the case of a server timeout, the SyncML client SHOULD establish a new HTTP session with the HTTP server and attempt to resend the current SyncML package, beginning with the first SyncML command for which the SyncML client has not received an acknowledgement. In the event that the SyncML client requested that no responses be sent, the SyncML client SHOULD begin retransmitting with the first SyncML command in the SyncML package.

In the case of a client timeout during a SyncML client-initiated data synchronization, the SyncML server SHOULD clean up the TCP connection and do no further processing of the SyncML request.

When using the SyncML Reference Toolkit, the TCP cleanup and drop of the physical connection can be requested by setting a parameter in the xptCloseCommunication() function.

5.2 Exchanging SyncML Messages

Once an HTTP connection has been established, one or more SyncML Message are transferred over the connection, by the SyncML client, in the entity body of HTTP requests or received from the server in the entity body of HTTP responses.

The POST method is used to transfer the SyncML message in an HTTP request.

The body of the HTTP request MUST always include a SyncML message. The content-type for the body MUST be the appropriate SyncML Message content type.

Generally, the Request-URI, in the start line, is the URI "path" component of the intended recipient resource for the request. The URI of the origin server is specified by the value in the Host request header.

The Request-URI can also be "*" to indicate that the request is intended for the origin server indicated by the absolute URI specified in the Host request header. In this case, the origin server would be the SyncML server. However, in general, the SyncML server will be a resource (e.g., servlet) under the origin server.

The HTTP response will always include the transport response status code. The body of the HTTP Response MUST always include a SyncML Message. The content-type for the body should be the appropriate SyncML message content type

5.2.1 Single Message Per Package

The following is a snippet of the minimal HTTP start line and request headers for a simple HTTP request where the body of the request is the clear-text, XML [5] SyncML message media type.

```
POST ./servlet/syncit HTTP/1.1
Host: www.datasync.org
Content-Type: application/vnd.syncml+xml; charset="utf-8"
Content-Length: 1023
Accept: application/vnd.syncml+xml
```

Were the body, the binary, WBXML [4] SyncML message media type, then no content encoding is necessary. HTTP does not support the Content-Transfer-Encoding of MIME, in any case.

The following is a snippet of the minimal HTTP for an example for a simple HTTP response where the body in the response is the clear-text, XML [5] SyncML message media type, as specified in the Accept request header in the HTTP request.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.syncml+wbxml; charset="utf-8"
Content-Length: 1023

-- HTTP body, represented in a format consistent with the SyncML content
type follows --
```

5.2.2 Multiple Messages Per Package

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

This version of the specification does NOT support transferring SyncML messages across HTTP using a "multipart" MIME media type.

5.3 Transport Commands

HTTP uses a number of types of commands. The following sections specify static conformance requirements for use of these commands in the SyncML HTTP binding.

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

Simple tables are used to specify this information

In these tables, optional features are specified by a "MAY", mandatory features are specified by a "MUST" and recommended features are specified by a "SHOULD", as defined in RFC 2119 [3].

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.

5.3.1 Methods

The following table summarizes the support for the HTTP methods in the SyncML binding.

Method	Client	Server
OPTIONS	MAY	MAY
GET	MAY	MAY
HEAD	MAY	MAY
POST	MUST	MUST
PUT	MAY	MAY
DELETE	MAY	MAY
TRACE	MAY	MAY
CONNECT	MAY	MAY

The SyncML client MUST use the `POST` or MAY use the `CONNECT` method (if supported) to send SyncML requests to the SyncML server. The `CONNECT` method is used to initiate a SSL session to authenticate the HTTP client to the HTTP server. A typical HTTP request start line would look like the following:

```
POST ./servlet/syncit HTTP/1.1
Host: http://www.syncml.host.com
```

In this example, the HTTP client is specifying a "./servlet/syncit" path component for the Request-URI, which is a particular SyncML resource on the HTTP server. The "Host" HTTP header field is needed to specify the absolute URI for the HTTP server.

The other HTTP methods MAY be supported by implementations conforming to this specification. However, they are not used at this time by the SyncML client.

5.3.2 General Headers

The following table summarizes the support for the HTTP general headers in the SyncML binding.

General Header	Client	Server
Cache-Control	MUST	MUST
Connection	MAY	MAY
Date	MAY	MAY
Pragma	MAY	MAY
Trailer	MAY	MAY
Transfer-Encoding	MUST	MUST
Upgrade	MAY	MAY
Via	MAY	MAY
Warning	MAY	MAY

The `Cache-Control` general header is used to force control over the caching mechanisms in the request/response chain between the HTTP client and the HTTP server. Implementations conforming to this specification MUST support this header. Use of this header with the `no-store` parameter will assure that SyncML messages sent by the SyncML client and SyncML server are not stored in cache storage. This will greatly assure that SyncML server and SyncML client are processing the messages sent by the SyncML clients and SyncML servers, respectively. The following is an example of the typical specification for this header:

```
Cache-Control: no-store
```

In addition, implementations conforming to this specification MUST support the `private` `Cache-Control` option to assure that responses do not get cached and possibly used by agents other than the SyncML client agent or the SyncML server agent.

```
Cache-Control: private
```

The `Transfer-Encoding` general header is used to indicate what (if any) type of transformation has been applied to the message body in order to safely transfer it between the sender and the recipient. Implementations conforming to this specification MUST support the `chunked` `Transfer-Encoding` option.

```
Transfer-Encoding: chunked
```

The other general headers MAY be supported by implementations conforming to this specification.

5.3.3 Request Headers

The following table summarizes the support for the HTTP request headers in the SyncML binding.

Request Header	Client	Server
Accept	MUST	MUST
Accept-Charset	MUST	MUST
Accept-Encoding	MAY	MAY
Accept-Language	MAY	MAY
Authorization	MAY	MAY
Expect	MAY	MAY
From	MAY	MAY
Host	MAY	MAY
If-Match	MAY	MAY
If-Modified-Since	MAY	MAY
If-None-Match	MAY	MAY
If-Range	MAY	MAY
If-Unmodified-Since	MAY	MAY
Max-Forwards	MAY	MAY
Proxy-Authorization	MAY	MAY
Range	MAY	MAY
Referer	MAY	MAY
TE	MAY	MAY
User-Agent	MUST	MUST

The `Accept` request header is used to specify which media types are acceptable in the response.

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.

The following is an example of how this header is specified to indicate that the client expects responses formatted according to the clear-text, XML [5] representation of SyncML data synchronization:

```
Accept: application/vnd.syncml+xml
```

The following is an example of how this header is specified to indicate that the client expects responses formatted according to the binary, WBXML [4] representation of SyncML device management

```
Accept: application/vnd.syncml.dm+wbxml
```

The Accept-Charset request header is used to specify which character sets are acceptable in the response. Server implementations conforming to this specification MUST support this header with the "UTF-8" character set value. Client implementations SHOULD support the "UTF-8" character set. Implementations MAY support additional, IANA registered character set values. Client implementations not supporting UTF-8 SHOULD take careful consideration of the potential impact of lack of UTF-8 support on interoperability of the device. If a recipient is unable to provide support for the character set encoding specified in the Accept-Charset request headers sent by the originator, the recipient MUST send to the originator a HTTP status 406, "Not acceptable". This is in keeping with [1]. Note that there will be no SyncML message sent with this response. The following is an example of how this header is specified to indicate that the client expects responses formated into the UTF-8 character set:

```
Accept-Charset: UTF-8
```

The `Authorization` request header is used by an HTTP client to authenticate itself to the HTTP server. Implementations conforming to this specification MAY support this header with the authorization values for "Basic" and "Digest Access Authentication" authentication schemes, as specified in [2]. The following is an example of how this header is specified to allow the HTTP client to authenticate itself with a Base64 character encoding of a userid of `Aladdin` and password of `open sesame`.

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

The `Proxy-Authorization` request header is similar to the `Authorization` header except that it is specified by the HTTP client and used only by the next proxy in the connection chain. Implementations conforming to this specification MAY support this header and with the authorization values for "Basic" and "Digest Access Authentication" authentication schemes, as specified in [2]. The following is an example of how this header is specified to allow the HTTP client to authenticate itself with a userid of `Aladdin` and password of `open sesame`.

```
Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

The `User-Agent` request header identifies the type of user agent originating the request. This information is useful for the HTTP server to provide automated recognition of user agents for the sake of tailoring responses to avoid particular limitations or to take advantage of special features in the HTTP client. Implementations conforming to this specification

MUST support this header and provide it in all HTTP requests. The following is an example of the usage of this header.

```
User-Agent: Foo Bar Sync Products v3.4
```

The other request headers MAY be supported by implementations conforming to this specification.

5.3.4 Response Headers

The following table summarizes the support for the HTTP response headers in the SyncML binding.

Response Header	Client	Server
Accept-Ranges	MAY	MAY
Age	MAY	MAY
Allow	MAY	MAY
Authentication-Info	MAY	MAY
Etag	MAY	MAY
Location	MAY	MAY
Proxy-Authenticate	MAY	MAY
Retry-After	MAY	MAY
Server	MAY	MAY
Vary	MAY	MAY
WWW-Authenticate	MAY	MAY

The `Authentication-Info` response header is defined by [2]. The header is used by an HTTP proxy or server to provide information back to the HTTP client about a successful HTTP client authentication. Implementations conforming to this specification MAY support this header with the Authentication-Information directives "`nextnonce`" and "`rspauth`". The former directive is used to specify the nonce to be used by the client for a future authentication. The nonce value SHOULD be a Base64 formatted string. The latter directive is used by the HTTP proxy or server to authenticate itself to the HTTP client. The following example shows how an HTTP server might use this response header to provide authentication credentials to an HTTP client that has successfully authenticated itself with the HTTP server.

```
Authentication-Info: nextnonce="Bruce"
rspauth="edd30630e82fabdc1e895d1d3a4c0450"
```

The `Proxy-Authenticate` response header is used by an HTTP proxy to challenge the authority of the HTTP client issuing it an HTTP request. Implementations conforming to this specification MAY support this header with the challenge values for "Basic" and "Digest

Access Authentication" authentication schemes and the "Realm", "Domain", "Nonce", "Stale" and "Algorithm" authentication parameters, as specified in [2]. The nonce value SHOULD be a Base64 formatted string. The "MD5" algorithm MAY be supported by implementations that conform to this specification. Other algorithms can also be supported. The following is an example of this header being used by an HTTP proxy to challenge a HTTP client with the `Digest` authentication scheme for the `http://www.syncml.host.com` realm.

```
Proxy-Authenticate: Digest Domain="http://www.datasync.org/servlet/syncit"
```

The `WWW-Authenticate` response header is used by the HTTP server to challenge the authority of the HTTP client issuing it an HTTP request. Implementations conforming to this specification MAY support this header with the challenge values for "Basic" and "Digest Access Authentication" authentication schemes and the "Realm", "Domain", "Nonce", "Stale" and "Algorithm" authentication parameters, as specified in [2]. The nonce value SHOULD be a Base64 formatted string. The "MD5" algorithm MAY be supported by implementations that conform to this specification. Other algorithms can also be supported. The following is an example of this header being used by an HTTP server to challenge an HTTP client with the `Basic` authentication scheme for the `WallyWord@syncml.host.com` realm.

```
WWW-Authenticate: Basic Realm="WallyWorld@syncml.host.com"
```

The other response headers MAY be supported by implementations conforming to this specification.

5.4 Security

HTTP client and HTTP server authentication is based on the mechanism defined in [2]. Implementations conforming to this implementation MUST support this mechanism for "Basic" and "Digest Access Authentication". The Base64 character encoded "Basic" and "MD5" algorithm of the "Digest Access Authentication" authentication schemes MAY be supported. The HTTP headers and parameters that MUST be supported are described in the previous sections for request and response headers.

The Content-MD5 header field can be used to provide a message integrity check of the SyncML Message in the body. Use of this header is a good idea for detecting accidental modification of the entity-body in transit, but is not proof against malicious attack.

HTTP proxy and HTTP server implementations conforming to this specification MAY support both the ability to challenge unauthenticated requests and also accept authentication request headers in a request; which will not require subsequent challenge responses unless some part of the credential is incorrect. The latter requirement is required to address the need for minimal request/response traffic for mobile networks.

The authentication mechanisms defined by [2] address protecting the authentication credentials. However, the remainder of the HTTP request and response messages are available to the eavesdropper. For more robust security for the HTTP connection, SSL or HTTPS SHOULD be used. These mechanisms are not mandated for implementations conforming to this specification.

6 Examples

The XML [5] in these examples are generally formatted one element type per line for readability purposes, but need not be formatted as such in actual usage.

The following is a more complex example of an HTTP request with an actual clear-text, XML SyncML message in the body.

```
POST ./servlet/syncit HTTP/1.1

Host: http://www.syncml.host.com
Accept: application/vnd.syncml+xml
Accept-Charset: utf-8
Accept-Encodings: chunked
Accept-Language: en-US
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Content-Type: application/vnd.syncml+xml; charset="utf-8"
User-Agent: Foo Bar Sync Products v3.4
Cache-Control: no-store
Transfer-Encoding: chunked

BD
<SyncML>
<SyncHdr>
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>1</SessionID>
<MsgID>1</MsgID>
<Target>
<LocURI>http://www.datasync.org/servlet/syncit</LocURI>
</Target>
B6
<Source>
<LocURI>IMEI:001004FF1234567</LocURI>
<LocName>Bruce1's Device</LocName>
</Source>
<Cred>
<Data>QWxhZGRpbjpvcGVuIHNlc2FtZQ==</Data>
</Cred>
</SyncHdr>
<SyncBody>
<Sync>
D2
<CmdID>1</CmdID>
<Target>
<LocURI>./mail/bruce1</LocURI>
</Target>
<Source>
<LocURI>./calendar</LocURI>
</Source>
<Meta><Type xmlns="syncml:metinf">text/x-vcalendar</Type></Meta>
<Update>
<CmdID>2</CmdID>
B3
<Item>
<Source>
<LocURILocURI>./1012</LocURILocURI>
</Source>
```

```
<Data>BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
DTSTART:20000522T133000Z
DTEND:20000522T143000Z
SUMMARY:Project Review
7A
CLASS:CONFIDENTIAL
CATEGORIES:APPOINTMENT
END:VEVENT
END:VCALENDAR
</Data>
</Item>
</Update>
</Sync>
<Final/>
</SyncBody>
</SyncML>
```

The following is a more complex example of an HTTP response with an actual clear-text, XML SyncML message in the body.

```
HTTP/1.1 200 OK
Authentication-Info: nextnonce="Bruce"
  rspauth="edd30630e82fabdc1e895d1d3a4c0450"
Content-Type: application/vnd.syncml+wbxml; charset="utf-8"
Content-Length: 1023

<SyncML>
<SyncHdr>
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>1</SessionID>
<MsgID>2</MsgID>
<Target>
<LocURI>IMEI:001004FF1234567</LocURI>
</Target>
<Source>
<LocURI>http://www.datasync.org/servlet/syncit</LocURI>
</Source>
<Cred>
<Data>QWxhZGRpbjpvcGVuIHNlc2FtZQ==</Data>
</Cred>
</SyncHdr>
<SyncBody>
<Status>
<MsgRef>1</MsgRef> >
<CmdRef>1</CmdRef>
<Cmd>Sync</Cmd>
<TargetRef>./mail/bruce1</TargetRef>
<SourceRef>./calendar</SourceRef>
<StatusItem>
<Data>200</Data>
</StatusItem>
</Status>
<Sync>
<CmdID>1</CmdID>
<Target>
<LocURI>./calendar</LocURI>
```

```
</Target>
<Source>
<LocURI>./mail/bruce1</LocURI></Source>
<Meta><Type xmlns="syncml:metinf">text/x-vcalendar</Type></Meta>
<Update>
<CmdID>2</CmdID>
<Item>
<Source>
<LocURI>./50442009</LocURI>
</Source>
<Data>BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
DTSTART:20000519T210000Z
DTEND:20000519T230000Z
SUMMARY:Debug Fest
CLASS:CONFIDENTIAL
CATEGORIES:APPOINTMENT
END:VEVENT
END:VCALENDAR
</Data>
</Item>
</Update>
</Sync>
<Final/>
</SyncBody>
</SyncML>
```

7 References

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

[2] HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, IETF.

[3] Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, IETF.

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

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

[6] SyncML Representation Protocol, SyncML.

[7] SyncML Synchronization Protocol, SyncML.

[8] IANA registered character sets, IANA.