Web Runtime API (WRAPI) – Push

Approved Version 1.0 – 23 Sep 2014

Open Mobile Alliance

OMA-TS-WRAPI_Push-V1_0-20140923-A

Contents

Tables

Figures

1. Scope

This specification defines an API exposing the enabler services provided by OMA Push to applications executing in Web Runtime environments.

2. References

2.1 Normative References

[GSM-SMS] "3GPP TS 23.040 V9.3.0 (2010-09): 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Technical realization of the Short Message Service (SMS) (Release 9)", Sept 2010, 3GPP. URL: http://www.3gpp.org/ftp/Specs/html-info/23040.htm

[Push-CAI] "Push Client - Application Interface", Open Mobile Alliance™, OMA-TS-PushCAI-V1_1, URL:http://www.openmobilealliance.org/

[Push-OTA] "Push Over The Air", , Open Mobile Alliance™, OMA-TS-PushOTA-V2_3, URL:http://www.openmobilealliance.org/

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed et al., November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt

[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", R. Fielding et. al, January 1999, URL:http://www.ietf.org/rfc/rfc2616.txt

[RFC3428] "Session Initiation Protocol (SIP) Extension for Instant Messaging", B. Campbell et al., December 2002. URL: http://www.ietf.org/rfc/rfc3428.txt

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee et al. January 2005. URL: http://tools.ietf.org/html/rfc3986

[RFC4627] "The application/json Media Type for JavaScript Object Notation (JSON)", D. Crockford, July 2006, URL:http://www.ietf.org/rfc/rfc4627.txt

[RFC5724] "URI Scheme for Global System for Mobile Communications (GSM) Short Message Service (SMS)", E. Wilde et. al, January 2010, URL:http://tools.ietf.org/rfc/rfc5724.txt

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/

[W3C-CORS] "Cross-Origin Resource Sharing",W3C, URL: http://www.w3.org/TR/cors/

[W3C-EventSource] "Server-Sent Events", W3C, URL: http://www.w3.org/TR/EventSource/

[W3C-FileAPI] "File API", W3C, URL: http://www.w3.org/TR/FileAPI/

[W3C-URLENC] W3C HTML 2.0 Specification, form-urlencoded Media Type, URL: http://www.w3.org/MarkUp/html-spec/html-spec_8.html#SEC8.2.1

[W3C-WARP] "Widget Access Request Policy", W3C, URI: http://www.w3.org/TR/widgets-access/

[WAC-2.0-Security] "WAC 2.0 – Widget Security and Privacy", Wholesale Application Community, 2011. URL: http://specs.wacapps.net/2.0/jun2011/

[WRAPI-API-Patterns] "Web Runtime API (WRAPI) – Design Patterns", Open Mobile Alliance™, OMA-TS-WRAPI_Design_Patterns-V1_0, URL:http://www.openmobilealliance.org/

[XMLSchema1] W3C Recommendation, XML Schema Part 1: Structures Second Edition, URL: http://www.w3.org/TR/xmlschema-1/

[XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes Second Edition, URL: http://www.w3.org/TR/xmlschema-2/

2.2 Informative References

[OMADICT] "Dictionary for OMA Specifications", Version 2.7, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_7, URL:http://www.openmobilealliance.org/

[OMNA] "OMA Naming Authority". Open Mobile Alliance™. URL: http://www.openmobilealliance.org/OMNA.aspx

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.

3.2 Definitions

API Patterns	Design guidelines and requirements for definition of APIs
ECMAScript	Use definition from [OMADICT].
JavaScript	Use definition from [OMADICT].
Push API Server	Software which implements and exposes the Push API.
Push Client	Device based software which optionally implements the Push API.
Push Gateway	Network based software which optionally implements the Push API.
Uniform Resource Identifier	Use definition from [OMADICT].
User Agent	Use definition from [OMADICT].
Web	The World Wide Web, a content and application framework based upon hypertext and related technologies, e.g. XML, JavaScript/ECMAScript, CSS, etc.
Web Application	An application designed using Web technologies.
Web IDL	An IDL language for Web application APIs
Web Runtime	Client software that supports the execution of Web Applications

3.3 Abbreviations

API	Application Programming Interface
EventSource	The EventSource API
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OMA	Open Mobile Alliance
REST	REpresentational State Transfer
SCR	Static Conformance Requirements
SMS	Short Message Service
TS	Technical Specification
UA	User Agent
UE	User Equipment
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAC	Wholesale Applications Community

Webapp	Web Application
WRAPI	The OMA Web Runtime API enabler
XML	eXtensible Markup Language
XSD	XML Schema Definition

4. Introduction

This specification defines an API exposing the event notification enabler services provided by OMA Push, GSM SMS, SIP MESSAGE, and other such text messaging services to applications executing in Web Runtime environments. This API is referred to in this document as the Push API. Software that exposes the Push API is referred to in this document as a Push API Server.

4.1 Version 1.0

Version 1.0 of the Push API specification addresses the following aspects:

- Basis of the Push API design in the W3C API "Server-Sent Events" [W3C-EventSource]
- Support for a subset of the features of the OMA "Push Client - Application Interface" specification [Push-CAI]:
  - Push-OTA bearer binding, at minimum supporting SMS-based connectionless Push
  - To reduce the complexity of the Push API for this release, the ability to select specific OMA Push bearers to activate is deferred to a future release.

This limited scope of supported OMA Push features enables the API to use the existing W3C-EventSource API definition, while opening up (at minimum) the most widely deployed OMA Push bearer (SMS) to a new class of client applications. If the underlying platform supports other Push-OTA bearers (e.g. OTA-HTTP, OTA-SIP, etc), SMS, and SIP MESSAGE, events from these sources can also be delivered through the Push API.

5. Overview of the Push API in the OMA Push Architecture

Web applications can support both online and offline use cases with access to the OMA Push enabler, and can use the OMA-standardized content types or application-specific content.

OMA Push enables the direct delivery of content in network contexts (point-to-point IP, SMS, SIP/IMS, and broadcast/multicast) and via methods (e.g. connectionless Push) that are typically unsupported by W3C-standard implementations. OMA Push can complement HTML5 Web APIs such as Server-Sent Events [W3C-EventSource] and Web Sockets, with these additional capabilities that are unsupported by the HTML5 APIs.

The Push API provides a bridge between Web applications executing in Web browsers or widget runtime environments (WRT), and the enabler services provided by OMA Push or SMS text messaging. The relationship of the Push API to the overall architectural elements in devices and the OMA Push architecture is illustrated below.

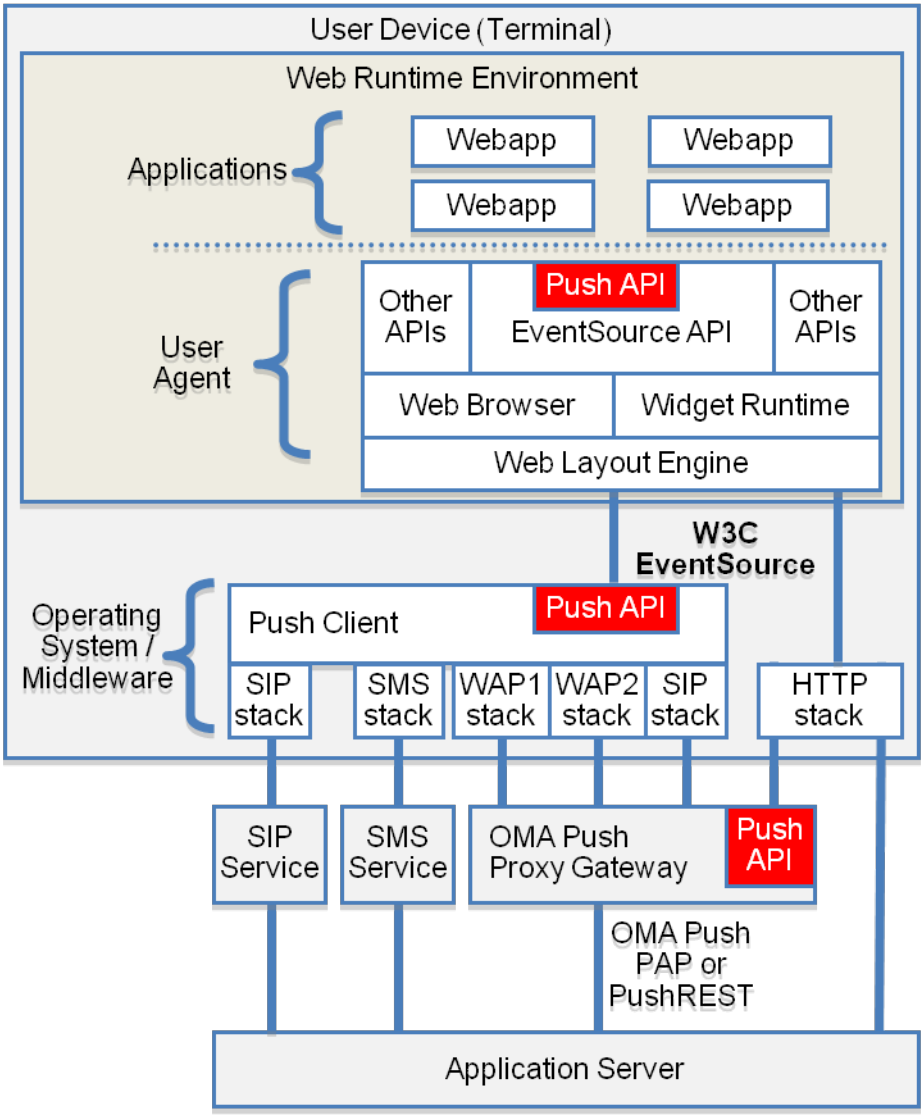

Figure 1: Relationship of Push API in the OMA Push Architecture

Three options are shown in the figure above for deployment of the Push API:

- As functionality of the Web User Agent (e.g. browser or Widget runtime): in this case the User Agent may be configured to locally serve EventSource connections to specific URLs, and take the necessary actions to deliver the requested events through the virtual EventSource connection.

- As functionality of an OMA Push Client in the device: in this case the Push Client acts as an EventSource server and provides the Push API extensions to the EventSource API, bridging the supported OMA Push protocols and text messaging enablers (e.g. GSM SMS and SIP MESSAGE) to an EventSource connection established between the User Agent and the Push Client.
- As functionality of a remote (network-based) Push Gateway: in this case the Push Gateway acts as an EventSource server and provides the Push API extensions to the EventSource API, bridging the supported OMA Push protocols and optionally SMS to an EventSource connection established between the User Agent and the Push Gateway. The Push Gateway may be implemented as functionality of an OMA Push Proxy Gateway, exposing OMA Push Access Protocol (PAP) or PushREST APIs to Application Servers, and optionally additional unspecified interfaces for plain text message delivery.

For definition of their requirements in support of the Push API, these implementation are referred to in the following sections as the Push API Server.

6. Design Basis in W3C APIs

The Push API design is based upon the EventSource interface as defined in the W3C API "Server-Sent Events" [W3C-EventSource]. This approach is intended to serve the key objectives:

- Simply developer adoption of the Push API by aligning the API design pattern with the conventions already established by W3C for EventSource
- Promote development of the Push API within open-source projects contributing to the mainstream Web browsers, e.g. Webkit and Mozilla
- Promote convergence of OMA Push with W3C Web APIs, e.g. through the inclusion of Push API extensions in W3C specifications

The EventSource interface provides the basic functionality of the Push API, which includes:

- Ability to create a new EventSource, which establishes a connection with a source for server-sent events
- Ability to receive events related to the overall status of an EventSource, including
    - Notification of successful opening of an EventSource connection
    - Notification of a new message from an EventSource
    - Notification of errors in an EventSource connection

As defined by W3C Server-Sent Events, the EventSource constructor takes a URL parameter expected to use the http URI scheme. The Push API uses the same design, while extending the use of the URL parameter to enable OMA Push features such as filtering events by source, application type, and content type.

The Push API adapts data from these new event sources to the text/event-stream MIME-type processing model defined for EventSource.

7. The Push Interface

The Push API is based upon the EventSource interface defined in [W3C-EventSource]. EventSource is used by Web applications (Webapps) to setup a persistent HTTP connection to an EventSource server, enabling asynchronous delivery of server-initiated events.

In addition to the Push API specific functions described in this document, Push API Servers MUST support operation as a server for EventSource connections as defined in [W3C-EventSource].

User Agents which support operation as a virtual Push API Server MUST be configurable to associate specific origins (domain and port of URLs) with the Push API virtual EventSource service.

Push Clients which support operation as a Push API Server MUST listen on TCP port 4035 for incoming EventSource connection requests, when Push API service is enabled. Note: the conditions under which Push API service is enabled in a device are unspecified, e.g. they may be related to user settings for OMA Push or notification services in general.

Push API Servers MUST support multiple EventSource connections.

Typically, the origin of the Webapp will be different from the origin of the Push API Server. For example, the Push API origin of a Push Client acting as Push API Server will be localhost: 4035 (4035 being the registered TCP port for OMA WAP2 Push). Similarly, a Push Gateway acting as a Push API Server will likely not have the same origin as the Webapp. Thus in order for the Push API to be accessed by the Webapp, Cross-Origin Resource Sharing [W3C-CORS] is used to authorize the User Agent to establish the cross-origin connection.

The URL of Push API Servers are expected to be discovered by the Webapp through unspecified application-specific procedures.

In addition to the specific requirements given in the following sections, User Agents MUST support all aspects of the EventSource interface for the new event sources made available through the Push API, with the following exception:

- When operating as a virtual Push API Server for connectionless bearers as event sources, the “reconnection time” and reconnection processing requirements of [W3C-EventSource] are not applicable.

7.1 Establishing a New EventSource for Push

The EventSource API defines a single parameter for new EventSource objects: a URL representing the source from which event reception should be initiated.

For the Push API, the URL parameter is used to select an EventSource API server which can forward received events to the application, via the EventSource API, with the following options for push event filtering:

- by event source, using the “push-accept-source” parameter of the EventSource URL to select a comma-separated list of acceptable event sources, optionally including
    - one or more SMS source addresses in the format “sms:sms-recipient” where sms-recipient is as defined by [RFC5724], which indicates a request for delivery of events from specific SMS addresses
    - one or more SIP source addresses in the format “sip:user@domain”, which indicates a request for delivery of events from specific SIP addresses
    - the OMNA-registered URN “urn:oma:xml:push”, which indicates a request for delivery of any OMA Push message received from the supported OMA Push bearers
    - other arbitrary source address values, enabling the extension of the Push API to other eventing or messaging systems or application-specific source addressing
- by OMA Push Application ID , using the “push-accept-application-id” parameter of the EventSource URL, to select a comma-separated list of acceptable push application ids
- by content (MIME) type , using the “push-accept-content-type” parameter of the EventSource URL, to select a comma-separated list of acceptable push content types

Applications can create multiple EventSource objects for delivery of Push events. This allows the application to choose which types of Push sources should be activated, in any desired combination.

The application can use these options to apply specific event handlers for the different event sources. For example, the application may expect SMS from specific source addresses to have a specific format for processing. OMA Push messages may also have specific processing and validation requirements, e.g. for filtering by the application based upon the Push Application Id, and to parse or validate the message data for different Push content types.

7.1.1 Processing a New Push API EventSource Request

As described in [W3C-EventSource], the EventSource(url) constructor takes one argument, url, which specifies the resource to which to connect.

User Agents which support operation as a virtual Push API Server MUST process new EventSource requests (creation of new EventSource objects via JavaScript) for the Push API, if the url parameter matches a pre-configured origin associated with the Push API virtual EventSource service.

When the EventSource() constructor is invoked, the User Agent supporting the Push API Server capability MUST run these steps:

1. If the url parameter is recognized by the User Agent as an EventSource URL at which it is configured to serve Push API requests, run these steps
    a. If the url parameter contains no “push-accept-source” parameter, set the push accept source filter to the value “*”, otherwise run these steps
        i. throw a SYNTAX_ERR exception if the “push-accept-source” parameter contains anything other than a comma-separated list of values formatted as one or more of: “sms-recipient” fields per [RFC5724], the value “urn:oma:xml:push”, or the value “*”
        ii. set the push accept source filter to the value of the “push-accept-source” parameter
    b. If the url parameter contains a “push-accept-application-id” parameter, set the push accept application id filter to the value of the “push-accept-application-id” parameter, otherwise set the push accept application id filter to null
    c. If the url parameter contains a “push-accept-content-type” parameter, set the push accept content type filter to the value of the “push-accept-content-type” parameter, otherwise set the push accept content type filter to null
    d. If the push accept source filter contains the value “*” or values formatted as “sms-recipient” fields as defined by [RFC2754], activate SMS event delivery to the EventSource connection
    e. If the push accept source filter contains the value “*” or values formatted as SIP addresses, activate SIP event delivery to the EventSource connection
    f. If the push accept source filter contains the value “*” or the value “urn:oma:xml:push”, activate OMA Push event delivery to the EventSource connection
2. If the url parameter is not recognized by the User Agent as an EventSource URL at which it is configured to serve Push API requests, establish a new EventSource as described in [W3C-EventSource].

Push Clients or Push Gateways which support operation as a Push API Server MUST run these steps upon reception of a GET request for a URL at which Push API service is provided

1. If the request url contains no “push-accept-source” parameter, set the push accept source filter to the value “*”, otherwise run these steps
    a. Return a 403 FORBIDDEN response and abort these steps if the “push-accept-source” parameter contains any unsupported source types, e.g. for a Push Gateway that does not support SMS message delivery
    b. Set the push accept source filter to the value of the “push-accept-source” parameter
2. If the request url contains a “push-accept-application-id” parameter, set the push accept application id filter to the value of the “push-accept-application-id” parameter, otherwise set the push accept application id filter to “*”
3. If the request url contains a “push-accept-content-type” parameter, set the push accept content type filter to the value of the “push-accept-content-type” parameter, otherwise set the push accept content type filter to “*”

4. If the push accept source filter contains the value "*" or values formatted as "sms-recipient" fields as defined by [RFC2754], activate SMS event delivery to the EventSource connection
5. If the push accept source filter contains the value "*" or values formatted as SIP addresses, activate SIP event delivery to the EventSource connection
6. If the push accept source filter contains the value "*" or the value "urn:oma:xml:push", activate OMA Push event delivery to the EventSource connection
7. Return a successful response to the User Agent as described in [W3C-EventSource], including an "Access-Control-Allow-Origin" header with the value of the "Origin:" header received in the GET request.

7.1.1.1 Establishing a New EventSource for SMS Events

Below is an example of how an application can create a new EventSource for SMS events using the Push API, via Push Clients that support operation as a Push API Server.

```
try {
  var es = new EventSource("http://localhost:4035/?push-accept-source=sms:+14255551212");
                                          // Expect Push API events from SMS source +14255551212
  es.onmessage = function (event) {       // Event handler
    // Handle then new event: the example below just shows presentation of the content
    ediv = document.getElementById('esdata');    // get HTML element where new content is to be displayed
    ediv.innerHTML = event.data;          // Display the new content
    };
  }
catch(e) {
  // Handle EventSource setup exception
  }
}
```

Table 1: Javascript example for establishing a new event source for SMS events

The example below shows an EventSource connection that is setup and used by an app for some time, then switched to a Push API connection when the app no longer needs or can maintain the data connection. The Push API is served directly by the User Agent (browser or Widget runtime), which provides the bridge to SMS event sources, and delivers the events to the application-defined EventSource event handler. The User Agent is pre-configured to recognize URLs with the origin localhost:4035 (WAP Push OTA-HTTP port) as a virtual Push API service address.

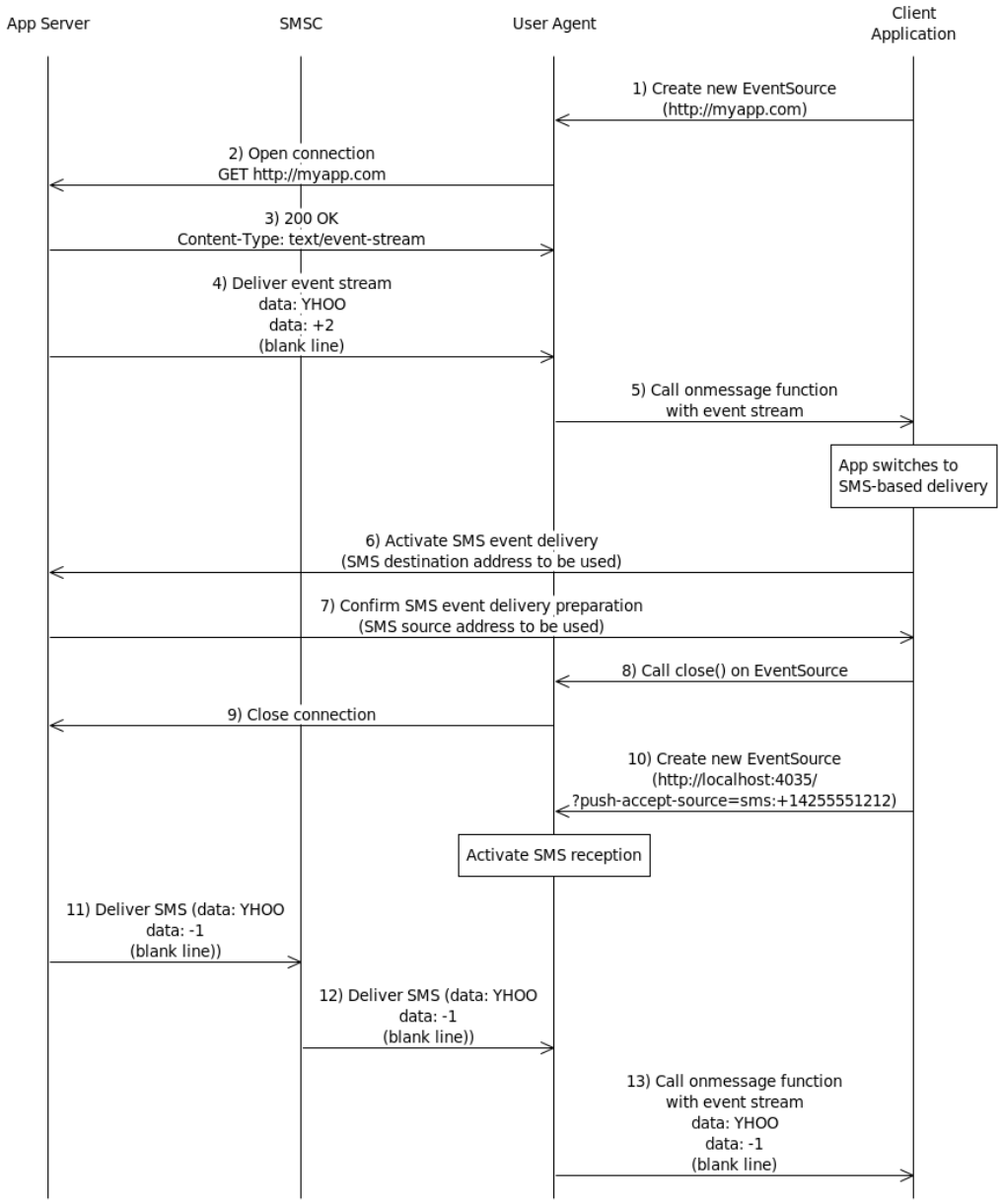

Figure 2: Switching from Connection-Based EventSource to SMS EventSource

1. Client App invokes the EventSource API with a URL meeting same-origin requirements.
2. The User Agent opens an EventSource connection to the server at the requested URL.
3. The App Server acks the opening of the EventSource connection.
4. The App Server delivers an event stream.
5. The User Agent calls the onmessage function defined for the EventSource object with the event stream.
6. Sometime later, the Client App decides to switch to connectionless event delivery via SMS. Coordination of the key parameters (e.g. SMS destination address and SMS source address to be used), and the trigger for switching to connectionless delivery (e.g. upon closure of the eventsource connection over HTTP), are assumed to occur at the application layer. The Client App initiates this coordination (in this example) by

requesting switch to connectionless delivery, and as needed providing SMS address information so the App Server knows where to send events.

7. The App Server confirms preparation of the switch to connectionless delivery, and provides its SMS source address so that the Client App can setup event source filtering.
8. The Client App calls the close() method on the eventsource object.
9. The User Agent closes the EventSource connection to the App Server.
10. Client App invokes the EventSource API with the URL for the local Push API server and includes a source filter parameter for the SMS address from which events should be received (http://localhost:4035/?push-accept-source=sms:+14255551212). Recognizing the EventSource URL origin localhost:4035 (WAP Push OTA-HTTP port) as a pre-configured virtual Push API service address, the User Agent activates SMS reception per the URL parameters.
11. The App Server has an event to be delivered, and notes that no EventSource connection is active to the Client App. The App Server delivers the event stream in an SMS message, either directly to the SMSC, or via an SMS API service.
12. The SMSC delivers the SMS message to the user's device.
13. The User Agent delivers the SMS message as an event stream, via the onmessage function defined for the EventSource object.

The example below shows an EventSource connection that is setup and used by an app for some time, then switched to a Push API connection when the app no longer needs or can maintain the data connection. The Push API is served directly by the User Agent (browser or Widget runtime), which provides the bridge to SMS event sources, and delivers the events to the application-defined EventSource event handler. The User Agent is pre-configured to recognize URLs with the origin localhost:4035 (WAP Push OTA-HTTP port) as a virtual Push API service address.

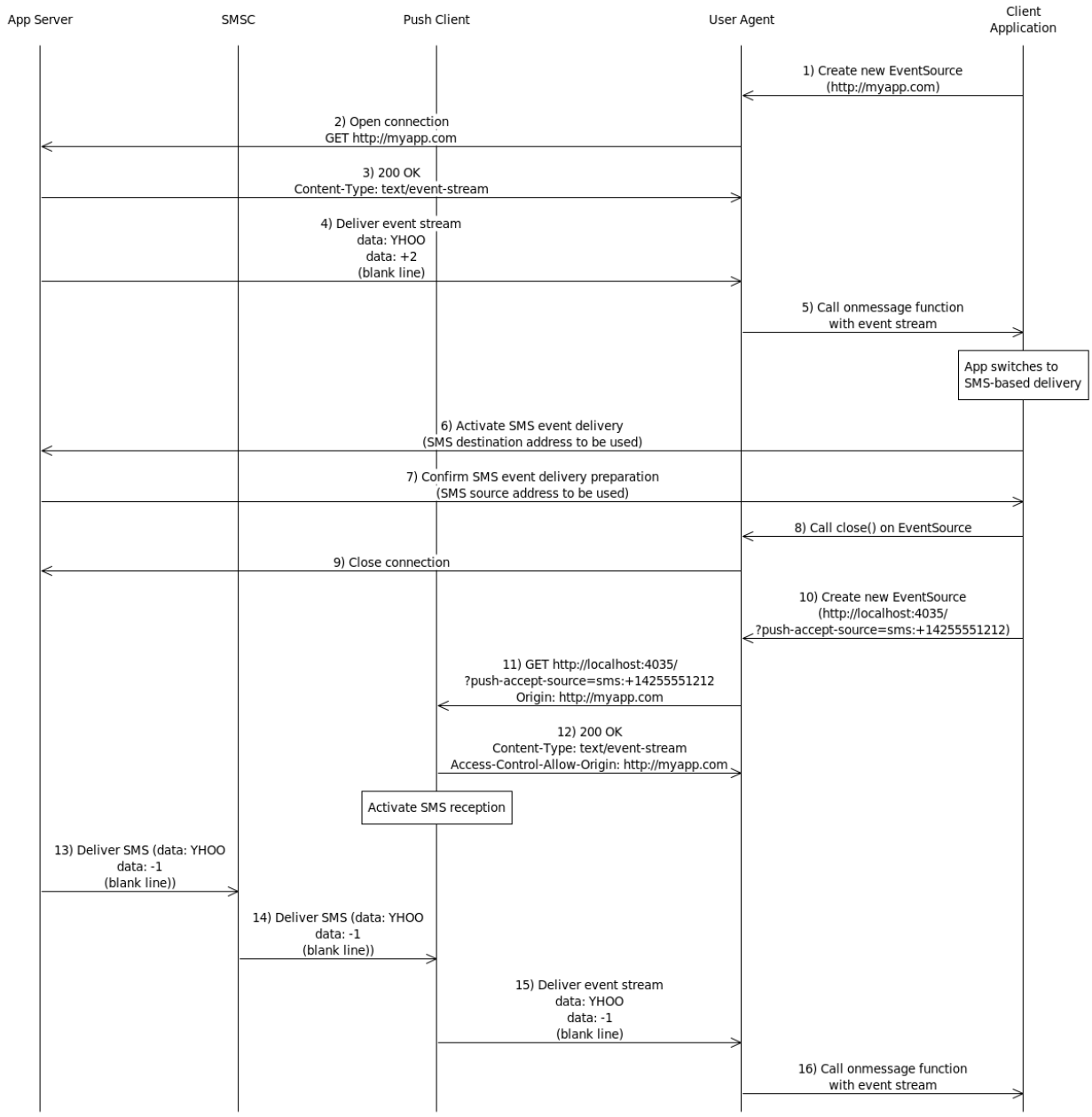

Figure 3: Switching from Connection-Based EventSource to SMS EventSource via Push Client

1. Application invokes the EventSource API with a URL meeting same-origin requirements.
2. The User Agent opens an EventSource connection to the server at the requested URL.
3. The App Server acks the opening of the EventSource connection.
4. The App Server delivers an event stream.
5. The User Agent calls the onmessage function defined for the EventSource object with the event stream.

6. Sometime later, the Client App decides to switch to connectionless event delivery via SMS. Coordination of the key parameters (e.g. SMS destination address and SMS source address to be used), and the trigger for switching to connectionless delivery (e.g. upon closure of the EventSource connection over HTTP), are assumed to occur at the application layer. The Client App initiates this coordination (in this example) by requesting switch to connectionless delivery, and providing its SMS destination address (so the App Server knows where to send events).
7. The App Server confirms preparation of the switch to connectionless delivery, and provides its SMS source address so that the Client App can establish the event source.
8. The Client App calls the close() method on the EventSource object.
9. The User Agent closes the EventSource connection to the App Server.
10. The Client App invokes the EventSource API with the URL for the local Push API server and includes a source filter parameter for the SMS address from which events should be received (http://localhost:4035/?push-accept-source=sms:+14255551212).
11. The User Agent establishes an EventSource connection to the Push Client at the URL.
12. The Push Client acks the EventSource connection setup, and establishes SMS reception if not already active.
13. The App Server has an event to be delivered, and notes that no EventSource connection is active to the Client App. The App Server delivers the event stream in an SMS message, either directly to the SMSC, or via an SMS API service.
14. The SMSC delivers the SMS message to the user's device.
15. The Push Client delivers the text of the SMS message via the EventSource connection.
16. The User Agent calls the onmessage function defined for the EventSource object with the event stream.

7.1.1.2 Establishing a New EventSource for OMA Push Events

The OMA Push Client functionality is assumed to be pre-configured in the device to establish connections over the supported bearers (e.g. OTA-WSP/SMS, OTA-WSP/IP, OTA-HTTP, OTA-SIP, etc) either through OMA Device Management or device-specific configuration mechanisms.

Below is an example of how an application can create a new EventSource for OMA Push events using the Push API. In this example, the application requests delivery of Service Indication messages only, and uses application logic to ignore all message except those from the expected source Push Initiator, using the X-Wap-Initiator-URI header.

```
try {
  var es = new EventSource('http://localhost:4035/?push-accept-source=urn:oma:xml:push&push-accept-
application-id=myapp.com/feed&push-accept-content-type=text/vnd.wap.si');
                                                // Expect Push API events from OMA Push sources
  es.onmessage = function (event) {             // Event handler
    // Handle the new event
    if (event.data.search(/X\-Wap\-Initiator\-URI: myapp.com/i)) {
                                                // Only accept from myapp.com
      var data = event.data.substring(event.data.indexOf('\n\n')+2);  // Data follows the first blank line
      parseSi(data);                                                  // Parse the SI content
      ediv = document.getElementById('esdata');                       // Get the output element
      ediv.innerHTML = pushText + "<br/><a href='" + pushUrl + "'>Click Here!</a>";  // Output the content
      }
    };
  }
catch(e) {
  // Handle EventSource setup exception
  }
}

var pushXml;       // Variable to hold XML DOM document created from the Push content
var pushUrl;       // Variable to hold the SI URL
var pushText;      // Variable to hold the SI text

function parseSi(data) {                                              // Parse SI content
  if (data.length > 0) {                                              // Ignore empty content (not expected)
    try {                                                             // Internet Explorer method
      pushXml=new ActiveXObject("Microsoft.XMLDOM");
      pushXml.async="false";
      pushXml.loadXML(data);
    }
    catch(e) {                                                        // Internet Explorer method failed
      try {                                                           // Try Mozilla etc (W3C) method
        var parser = new DOMParser();
        pushXml = parser.parseFromString(data, "text/xml");
      }
      catch(e1) {
        alert('Unable to parse content from EventSource server');     // Error in content
        return(false);
      }
    }
    var el = pushXml.getElementsByTagName("indication");              // Find <indication> element
    pushUrl = el[0].attributes["href"].value;                         // Get SI URL (href attribute)
    pushText = el[0].textContent;                                     // Get SI text
    return(true);
  }
  else {
    alert('No content from EventSource server');
    return(false);
  }
}
```

Table 2: Javascript example for establishing a new event source for OMA Push and processing received events

The example below shows an EventSource connection that is setup and used by an app for some time, then switched to a Push API connection when the app no longer needs or can maintain the data connection. The Push API is served by an OMA Push Client on the device. The Push Client provides the bridge to OMA Push event sources, and delivers the events via a local EventSource connection with the User Agent (browser or widget runtime). The Push Client is listening on the localhost address at port 4035 (WAP Push OTA-HTTP port).

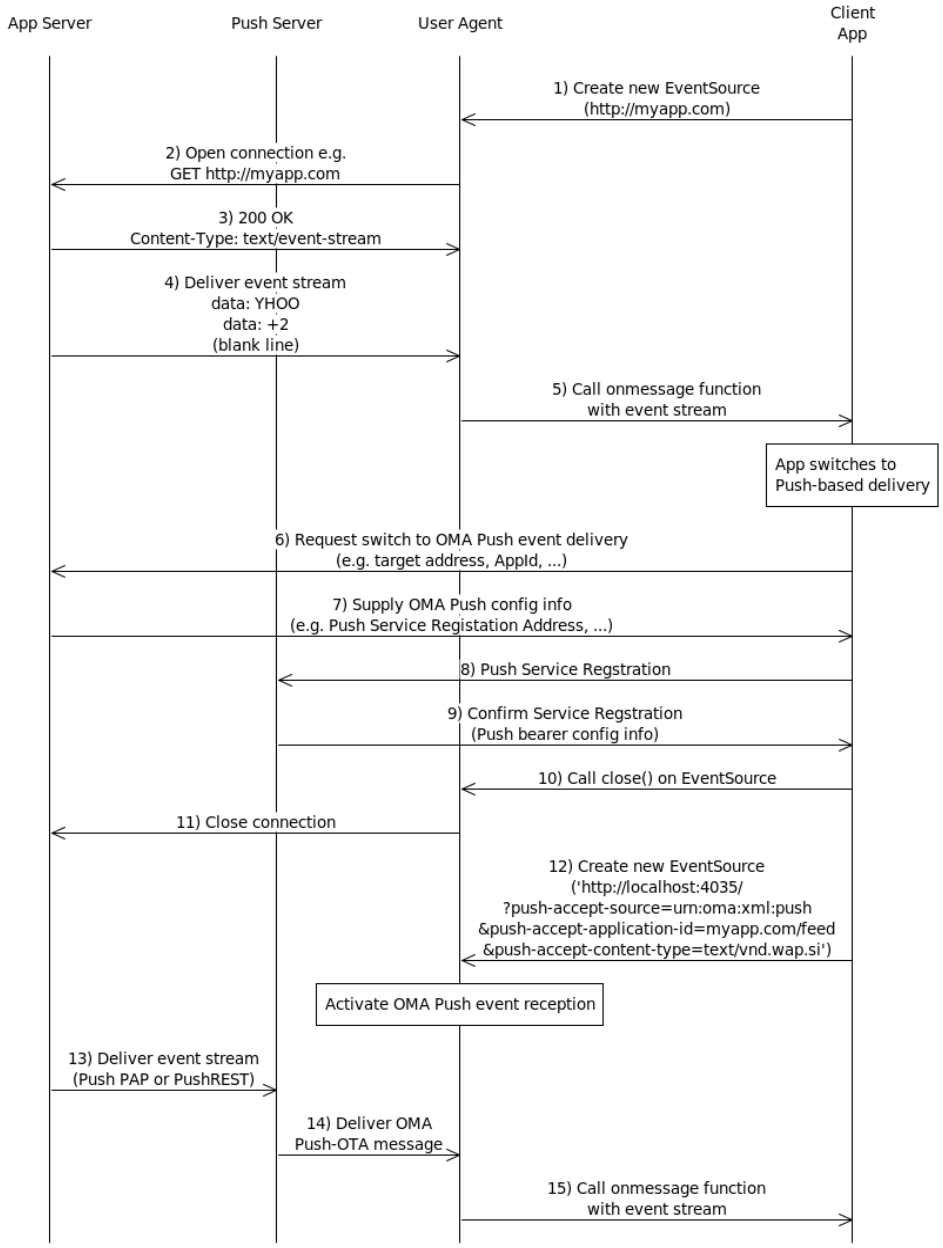

Figure 4: Switching from Connection-Based EventSource to OMA Push EventSource

1. Client App invokes the EventSource API with a URL meeting same-origin requirements.
2. The User Agent opens an EventSource connection to the server at the requested URL.
3. The App Server acks the opening of the EventSource connection.
4. The App Server delivers an event stream.
5. The User Agent calls the onmessage function defined for the EventSource object with the event stream.
6. Sometime later, the Client App decides to switch to event delivery via OMA Push. Coordination of the key parameters (e.g. OMA Push address, Push AppId, etc), and the trigger for switching to connectionless delivery (e.g. upon closure of the eventsource connection over HTTP), are assumed to occur at the application layer.

The Client App initiates this coordination (in this example) by requesting switch to OMA Push delivery, and providing its OMA Push address and AppId (so the App Server knows where to send events, and what AppId to use).

7. The App Server provides info that the Client App needs to complete the switch to OMA Push delivery, including a Push Service Registration address (where e.g. as described in [Push-OTA], the Client App and Push Server can coordinate establishment of Push service)
8. The Client App invokes the Push Service Registration procedure as described in [Push-OTA], providing an XML document requesting the configuration of Push service.
9. The Push Server confirms the Push service registration, and provides info needed by the Client App for client-side configuration, e.g. the Push API Server address URL and parameters.
10. The Client App calls the close() method on the eventsource object.
11. The User Agent closes the EventSource connection to the App Server.
12. The Client App invokes the EventSource API with a URL (http://localhost:4035/?push-accept-source=urn:oma:xml:push&push-accept-application-id=myapp.com/feed&push-accept-content-type=text/vnd.wap.si) for the local Push API server, including a source filter parameter for OMA Push events, a filter parameter for its application id, and a filter parameter for the expected content type. Recognizing the EventSource URL origin localhost: 4035 (WAP Push OTA-HTTP port) as a pre-configured virtual Push API service address, the User Agent activates OMA Push event reception per the URL parameters.
13. The App Server has an event to be delivered, and notes that no EventSource connection is active to the Client App. The App Server delivers the event stream in an OMA Push message, either using OMA Push PAP, PushREST, or via some other OMA Push API service.
14. The Push Server delivers the Push message message to the user's device.
15. The User Agent de-tokenizes the Push message (if required), and delivers the message headers and body as an event stream to the onmessage function defined for the EventSource object.

The example below shows an EventSource connection that is setup and used by an app for some time, then switched to a Push API connection when the app no longer needs or can maintain the data connection. The Push API is served by an OMA Push Client on the device. The Push Client provides the bridge to OMA Push event sources, and delivers the events via a local EventSource connection with the User Agent (browser or widget runtime). The Push Client is listening on the localhost address at port 4035 (WAP Push OTA-HTTP port).

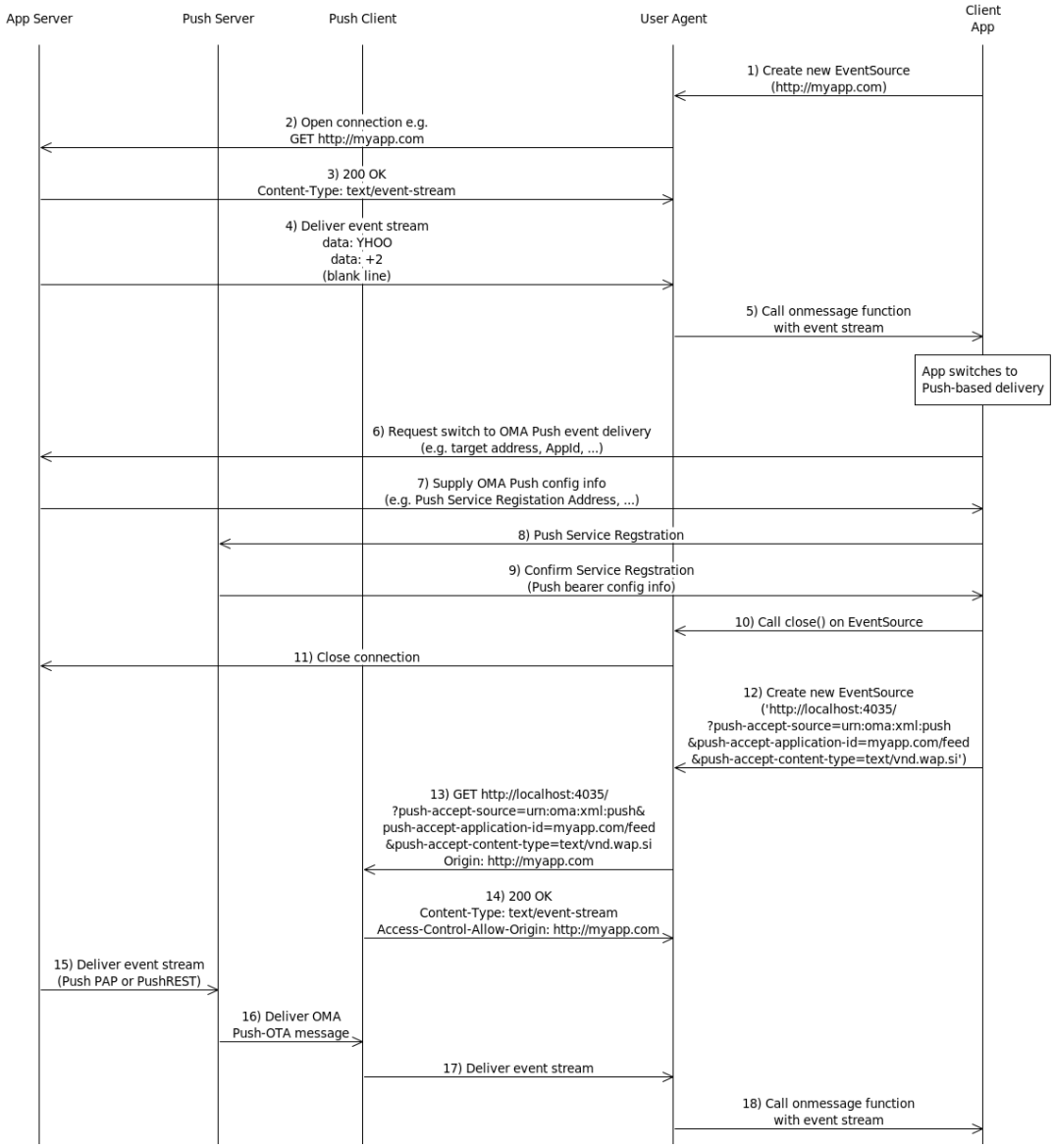

Figure 5: Switching from Connection-Based EventSource to OMA Push EventSource via Push Client

1. Client App invokes the EventSource API with a URL meeting same-origin requirements.
2. The User Agent opens an EventSource connection to the server at the requested URL.
3. The App Server acks the opening of the EventSource connection.
4. The App Server delivers an event stream.
5. The User Agent calls the onmessage function defined for the EventSource object with the event stream.
6. Sometime later, the Client App decides to switch to event delivery via OMA Push. Coordination of the key parameters (e.g. OMA Push address, Push AppId, etc), and the trigger for switching to connectionless delivery (e.g. upon closure of the EventSource connection over HTTP), are assumed to occur at the application layer. The

Client App initiates this coordination (in this example) by requesting switch to OMA Push delivery, and providing any initial information needed by the App Server (e.g. its OMA Push target address and AppId, so the App Server knows where to send events, and what AppId to use).

7. The App Server provides info that the Client App needs to complete the switch to OMA Push delivery, including a Push Service Registration address (where e.g. as described in OMA Push 2.3 Push-OTA, the Client App and Push Server can coordinate establishment of Push service)
8. The Client App invokes the Push Service Registration procedure as described in OMA Push 2.3 Push-OTA, providing an XML document requesting the configuration of Push service.
9. The Push Server confirms the Push service registration, and provides info needed by the Client App for client-side configuration, e.g. the Push API Server address URL and parameters.
10. The Client App calls the close() method on the EventSource object.
11. The User Agent closes the EventSource connection to the App Server.
12. The Client App invokes the EventSource API with a URL (http://localhost:4035/?push-accept-source=urn:oma:xml:push&push-accept-application-id=myapp.com/feed&push-accept-content-type=text/vnd.wap.si) for the local Push API server, including a source filter parameter for OMA Push events, a filter parameter for its application id, and a filter parameter for the expected content type.
13. The User Agent opens an EventSource connection to the Push Client at the requested URL. The User Agent includes the Origin header as this is a cross-origin resource request.
14. The Push Client acks the EventSource connection setup, including the Access-Control-Allow-Origin header authorizing the User Agent to use the cross-origin resource, and activates OMA Push reception event if not already active.
15. The App Server has an event to be delivered, and notes that no EventSource connection is active to the Client App. The App Server delivers the event stream in an OMA Push message, either using OMA Push PAP, PushREST, or via some other OMA Push API service.
16. The Push Server delivers the Push message to the Push Client on the user's device.
17. The Push Client de-tokenizes the Push message (if required), and delivers the message headers and body via the EventSource connection.
18. The User Agent calls the onmessage function defined for the EventSource object with the event stream.

The example below shows an EventSource connection that is setup and used by an app through a Push Client acting as a local Push API Server. At some point determined by the Push Client or App Server, a switch is made to use of OMA Push and SMS as event delivery bearers. The switch occurs transparently to the Client App. In this case in addition to acting as a bridge to OMA Push and SMS event sources, the Push Client acts as a local proxy for remote EventSource connections, and delivers received events via a local EventSource connection with the User Agent (browser or widget runtime). The Push Client is listening on the localhost address at port 4035 (WAP Push OTA-HTTP port).

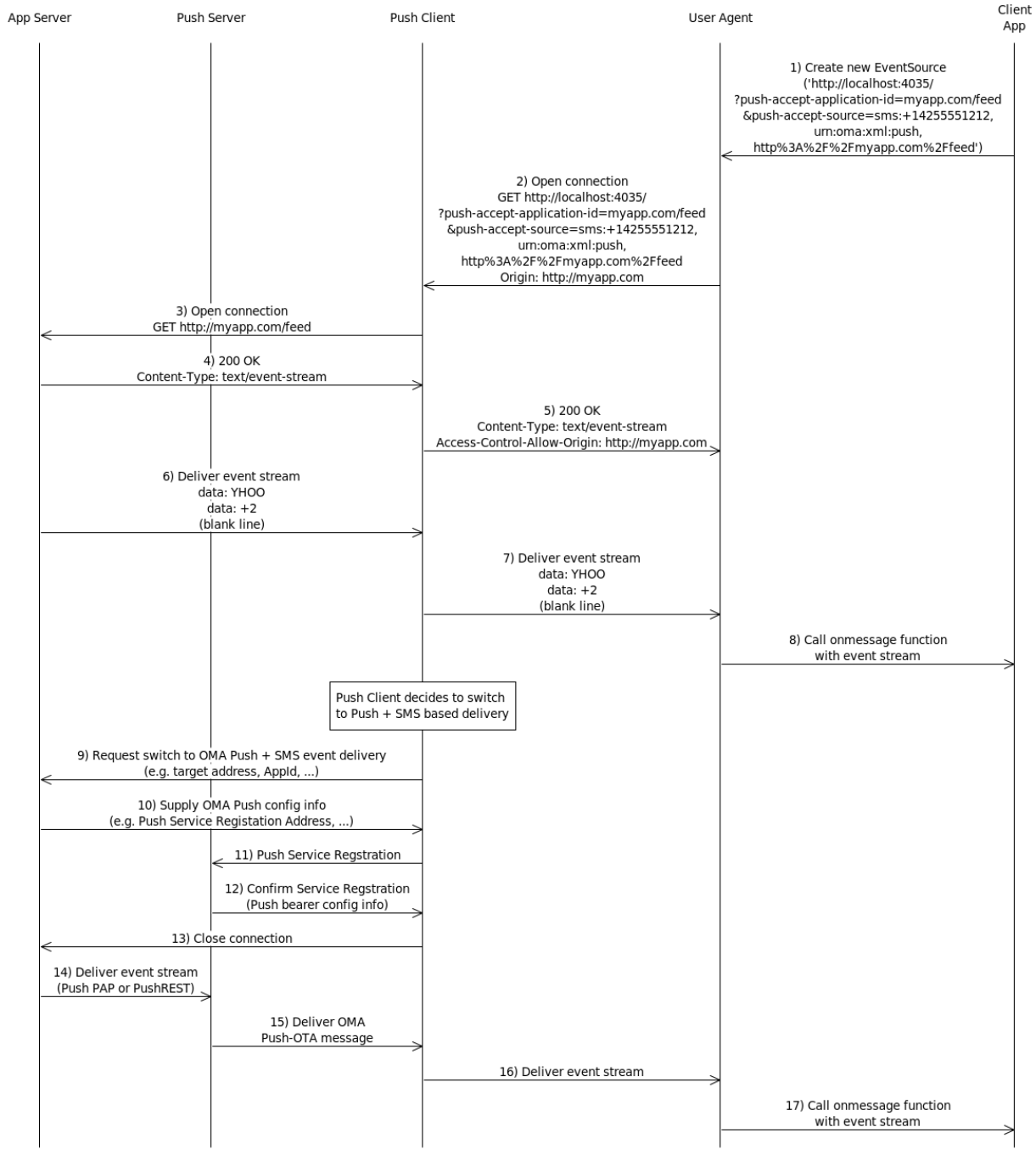

Figure 6: Seamless Switching from Connection-Based EventSource to OMA Push + SMS EventSource via Push Client

1. The Client App invokes the EventSource API with the URL for the local Push API server and includes a filter parameter for its application id, and source filter parameters for OMA Push events, SMS events, and an encoded

URI representing a request for a proxy EventSource connection. In this example, the ability to proxy EventSource connections as part of the Push API is an implementation extension.
2. The User Agent opens an EventSource connection to the Push Client at the requested URL. The User Agent includes the Origin header as this is a cross-origin resource request.
3. The Push Client recognizes the encoded URI in the accept source parameter as a request for proxy EventSource connection, and opens an EventSource connection to the App Server
4. The App Server acks the opening of the EventSource connection.
5. The Push Client acks the EventSource connection setup, including the Access-Control-Allow-Origin header authorizing the User Agent to use the cross-origin resource, and activates OMA Push reception event if not already active.
6. The App Server delivers an event stream.
7. The Push Client delivers the event stream.
8. The User Agent calls the onmessage function defined for the EventSource object with the event stream.
9. Sometime later, the Push Client decides to switch to drop the EventSource connection and continue reception via OMA Push and SMS. The conditions under which this switch occurs (and possible coordination with the App Server) are assumed to be implementation-specific. The Push Client initiates this coordination (in this example) by requesting switch to OMA Push and SMS delivery, and providing any initial information needed by the App Server (e.g. its OMA Push target address and AppId, so the App Server knows where to send events, and what AppId to use).
10. The App Server provides info that the Push Client needs to complete the switch to OMA Push delivery, including a Push Service Registration address (where e.g. as described in [Push-OTA], the Push Client and Push Server can coordinate establishment of Push service)
11. The Push Client invokes the Push Service Registration procedure as described in [Push-OTA], providing an XML document requesting the configuration of Push service.
12. The Push Server confirms the Push service registration, and provides info needed by the Push Client for client-side configuration, e.g. the Push API Server address URL and parameters.
13. The Push Client closes the EventSource connection to the App Server.
14. The App Server has an event to be delivered, and notes that no EventSource connection is active. The App Server delivers the event stream in an OMA Push message, either using OMA Push PAP, PushREST, or via some other OMA Push API service.
15. The Push Server delivers the Push message message to the Push Client on the user's device.
16. The Push Client de-tokenizes the Push message (if required), and delivers the message headers and body via the EventSource connection.
17. The User Agent calls the onmessage function defined for the EventSource object with the event stream.

7.2 Applying Filters on Push Events

The Push API provides for three types of event filters, which are referred to above as the push accept source filter, push accept application id filter, and the push accept content type filter. Each of these filters can be specified by the Webapp in the EventSource URL, as a comma-separated list of values.

The Push API Server MUST deliver only events that match the push accept source filter, push accept application id filter, and push accept content type filter.

When applying the filters, the Push API Server MUST run these steps:

1. If the event is an OMA Push event, run these steps
    a. If the push accept application id filter does not contain the value “urn:oma:xml:push”, abort these steps and do not deliver the event via the EventSource connection.
    b. If the push accept application id filter has the value “*” or the received “X-WAP-Application-ID:” header is a match for any value in the push accept application id filter, skip to step 1d.
    c. Abort these steps and do not deliver the event via the EventSource connection.
    d. If the push accept content type filter has the value “*” or the received “Content-Type:” header, minus any MIME type parameter field [RFC2045], is a match for any value in the push accept content type filter, skip to step 1f.
    e. Abort these steps and do not deliver the event via the EventSource connection.
    f. Deliver the event via the EventSource connection.
2. If the event is from the SMS bearer, received in a SIP MESSAGE, or received from some other plain text message facility, run these steps

a. If the event was received from a GSM SMS bearer, set the source address to the value of the TP-Originating-Address field of the SMS message [GSM-SMS].
b. If the event was received from a SIP bearer in a SIP MESSAGE, set the source address to the value of the SIP URI in the "From:" header of the SIP MESSAGE [RFC3428].
c. If the event was received from another (unspecified) text messaging facility, set the source address to the relevant value.
d. If the source address matches any value in the push accept source filter, skip to step 2f
e. Abort these steps and do not deliver the event via the EventSource connection
f. Deliver the event via the EventSource connection

7.3 Mapping of Events to the text/event-stream MIME type

EventSource is designed for delivery of a particular message format in event streams, per the text/event-stream MIME type processing model defined in [W3C-EventSource]. To enable developers to use a consistent approach to accessing event data, the typical structure of OMA Push-OTA events (headers+body) is mapped to the EventSource event-stream format.

For received OMA Push messages with a Content-Type header with value "text/event-stream", Push API Servers MUST interpret and deliver the received message body as an event stream, as described in [W3C-EventSource] section 7 "Interpreting an event stream".

For received OMA Push messages with Content-Type header set to values other than "text/event-stream", Push API Servers MUST deliver a single EventSource event for each set of entity-headers or entity-body in the message body:

- The entity-headers, if any
- The entity-body, if any

Push API Servers MUST dispatch the individual events in the order that the event data occurred in the Push message.

For entity-headers, Push API Servers must deliver each set of entity-headers as an EventSource event, as follows:

- Setting the event field to the string "headers"
- Setting the data field to a text string concatenating all Push message headers, using the following processing model:
    - if a Content-Type header for a OMA-defined compressed MIME type (e.g. application/vnd.oma.sic) is included in the entity-headers, set the Content-Type header to the equivalent uncompressed MIME type (e.g. application/vnd.oma.si).
    - For each message header, add the message header followed by a U+000A LINE FEED (LF) character to the data buffer
- Output the event and data field, followed by a blank line, to the EventSource connection.

For entity-bodies, Push API Servers must deliver each entity-body as an EventSource event, as follows:

- Setting the event field to the string "message"
- Setting the data field to a text string containing the Push message content, using the following processing model:
    - if the message body has a OMA-defined compressed MIME type (e.g. application/vnd.oma.sic), decompress the message headers and body
    - For each line of the message body, add the line followed by a single U+000A LINE FEED (LF) character to the data buffer
- Output the event and data field, followed by a blank line, to the EventSource connection.

For received messages from sources other than OMA Push, Push API Servers MUST deliver a single EventSource event, as follows:

- Setting the event stream "event" field to a string identifying the source type. In particular:
    - set the event field to the string "SMS"in case the event was received from a GSM SMS bearer,
    - set the event field to the string "SIP" in case the event was received from a SIP bearer in a SIP MESSAGE,

    - set the event field to another string based on other implementation-specific source type.
- Setting the data field to the event text content.
- Output the event and data field, followed by a blank line, to the EventSource connection.

Note: to prevent EventSource event delivery errors, messages delivered through EventSource connections cannot contain blank lines, as per the EventSource text/event-stream processing model, that will cause an event to be dispatched with an incomplete data buffer.

7.4 Terminating an EventSource for Push

When the close() method is invoked on an EventSource object, User Agents which support operation as a virtual Push API Server MUST terminate delivery of SMS and WAP Push message events to the EventSource object if applicable.

Push Clients or Push Gateways which support operation as a Push API Server MUST terminate delivery of SMS and WAP Push message events to the EventSource connection when a request to close the EventSource connection is received.

8. Security Considerations

The API defined in this specification can be used to access incoming SMS messages and OMA Push events. This may result in the disclosure of information related to a user's contacts, other applications that use SMS or OMA Push, and other personally identifying information carried by these event sources. The distribution of this information could potentially compromise the user's privacy, or the user's contacts' privacy. A conforming implementation of this specification must provide a mechanism that protects the user's privacy and this mechanism should ensure that no SMS or OMA Push event data is accessible without the user's express permission.

Other than the capabilities described in the following sections, the host device may have additional security-enhancing capabilities such as support for the OMA Push Whitelist feature, or other SMS filtering, spam control, or content filtering capabilities. Regardless, developers need to use caution in processing events from potentially unknown sources.

8.1 Restricting Access to Local Push API Service

Because Push Clients that implement a local Push API service listen on TCP port 4035 for incoming EventSource connections, to protect the device from intrusion attacks, only local requests may be served.

Push Clients MUST only accept incoming EventSource connections originated in the same device, i.e. for which the source address is the device’s own IP address.

8.2 Push API and the Same-Origin Policy

As defined by [W3C-EventSource], security of the EventSource interface for HTTP-based resources is based upon the standard “same-origin policy” security design of the Web. The same-origin policy is applicable to browser context applications, for which a specific origin server can be determined, so EventSource data from the same origin can also be trusted under the same-origin policy. The user thus implicitly chooses to trust the application by browsing to it at the origin server resource address (Web page that starts the application).

Prearranged trust relationships, or explicit user consent may apply in other contexts. For widget context applications, the ablity to access network resources including via EventSource is based upon declaration of the network domains that the application intends to access, in the widget configuration document as described by [W3C-WARP] for HTTP-based resources, and by additional security framework capabilities for non-HTTP-based resources (e.g. inclusion of API feature URIs in the widget configuration document per [WAC-2.0-Security]). For widgets, during widget installation the user is typically informed which APIs and resources the widget has declared a need to access. The user is able to provide consent at that time, or later e.g. if an applicable security policy requires a user prompt for each application session or API use.

For the Push API, the cross-origin request processing of EventSource provides the necessary security protection for the user. Since the origin of the Webapp is expected to be different from the origin of the Push API Server as described in section 7, in order for the Push API to be accessed by the Webapp, Cross-Origin Resource Sharing [W3C-CORS] is used to authorize the User Agent to establish a the cross-origin connection to the Push API server. Mechanisms for managing trust of specific origins for access to Push API services provided by Push Clients or User Agents directly are unspecified, but MAY include user prompts or pre-arranged trust relationships.

User Agents that support Push API service access via a Push Client or Push Gateway MUST support the establishment of cross-origin EventSource connections.

If a Push Client or Push Gateway acting as a Push API Server accepts the cross-origin connection for a Push API request, it MUST indicate acceptance the request through the “Access-Control-Allow-Origin:” header as described in section 7. Note that if a User Agent acting as a virtual Push API Server accepts the cross-origin connection for a Push API request, its response is implicit in the successful return of a new EventSource object.

8.3 Privacy considerations for implementors of the Push API

Push Clients and User Agents MUST NOT provide Push API event source access to Webapps without the express permission of the user. Push Clients and User Agents MUST acquire consent for permission through a user interface, unless a prearranged trust relationship applies. Such user interfaces may be implemented in various ways, e.g. through a popup dialog which is presented by the Push Client or User Agent when a Webapp requests Push API access, through a system setting related to Push API services, or other general privacy-related system settings.

The user interface for consent MUST include the Webapp's origin. Those permissions that are acquired through the user interface and that are preserved beyond the current browsing session (i.e. beyond the time when the browsing context is navigated to another URL outside the Webapp, or the local EventSource connection is closed) MUST be revocable and, the Push Client or User Agent MUST respect revoked permissions.

Obtaining the user's express permission to access one Push API source does not imply the user has granted permission for the same application to access other sources provided by the Push API, as part of the same permission context. If a user has expressed permission for an implementation to, e.g. access incoming SMS events from a particular SMS source address, the implementation MUST seek the user's express permission if and when any additional event sources are accessed via the Push API.

Since they do not have the opportunity to present an authorization dialog to the user during Push API access establishment, Push Gateways SHOULD provide some alternate means for user pre-authorization of Webapp access to Push API services. For example, the Push Gateway can use a Push Service Indication to deliver an authorization message and confirmation link to the user's browser, which when selected confirms that the user authorizes the specific Webapp to access Push services.

User Agents, Push Clients, or Push Gateways MAY support prearranged trust relationships that do not require such user interfaces or per-request confirmation processes.

8.4 Application Security

For SMS, SIP, and other text message event sources, application security is provided by explicit authorization of event sources. It is assumed that the application has gained the necessary degree of trust in the content provided by an event source, prior to initiating event reception from that source.

For OMA Push event sources, applications generally can depend upon the security of the OMA Push enabler as typically deployed. However defensive programming measures should always be applied by the developer, to ensure that the XML-based content and other content provided through OMA Push is safely processable.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-WRAPI_Push-V1_0-20140923-A	23 Sep 2014	Status changed to Approved by TP TP Ref # OMA-TP-2014-0216-INP_WRAPI_V1_0_ERP_for_final_Approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for User Agent

Item	Function	Reference	Requirement
WRAPI-PUSH-UA-001-M	Support all applicable EventSource functions	7	
WRAPI-PUSH-UA-002-M	Support configuration and operation as virtual Push API Server	7	
WRAPI-PUSH-UA-003-M	Support multiple EventSource objects	7	
WRAPI-PUSH-UA-004-M	Processing a new EventSource request	7.1.1	
WRAPI-PUSH-UA-005-M	Applying event filters	7.2	
WRAPI-PUSH-UA-006-M	Mapping of events to the text/event-stream MIME type	7.3	
WRAPI-PUSH-UA-007-M	Cross-origin EventSource connections	8.1	
WRAPI-PUSH-UA-008-M	Privacy considerations	8.3	

B.2 SCR for Push Client

Item	Function	Reference	Requirement
WRAPI-PUSH-C-001-M	Support all applicable EventSource functions	7	
WRAPI-PUSH-C-002-M	Serve EventSource connections on localhost:4035 as local Push API server.	7	
WRAPI-PUSH-C-003-M	Support multiple EventSource objects	7	
WRAPI-PUSH-C-004-M	Processing a new EventSource request	7.1.1	
WRAPI-PUSH-C-005-M	Applying event filters	7.2	
WRAPI-PUSH-C-006-M	Mapping of events to the text/event-stream MIME type	7.3	
WRAPI-PUSH-C-007-M	Resticting access to local Push API service	8.1	
WRAPI-PUSH-C-008-M	Cross-origin EventSource connections	8.1	
WRAPI-PUSH-C-009-M	Privacy considerations	8.3	

B.3 SCR for Push Gateway

Item	Function	Reference	Requirement
WRAPI-PUSH-S-001-M	Support all applicable EventSource functions	7	
WRAPI-PUSH-S-002-M	Serve EventSource connections for Push API service.	7	
WRAPI-PUSH-S-003-M	Support multiple EventSource objects	7	
WRAPI-PUSH-S-004-M	Processing a new EventSource request	7.1.1	
WRAPI-PUSH-S-005-M	Applying event filters	7.2	
WRAPI-PUSH-S-006-M	Mapping of events to the text/event-stream MIME type	7.3	
WRAPI-PUSH-S-007-M	Cross-origin EventSource connections	8.1	
WRAPI-PUSH-S-008-M	Privacy considerations	8.3	

Appendix C. Push API Usage

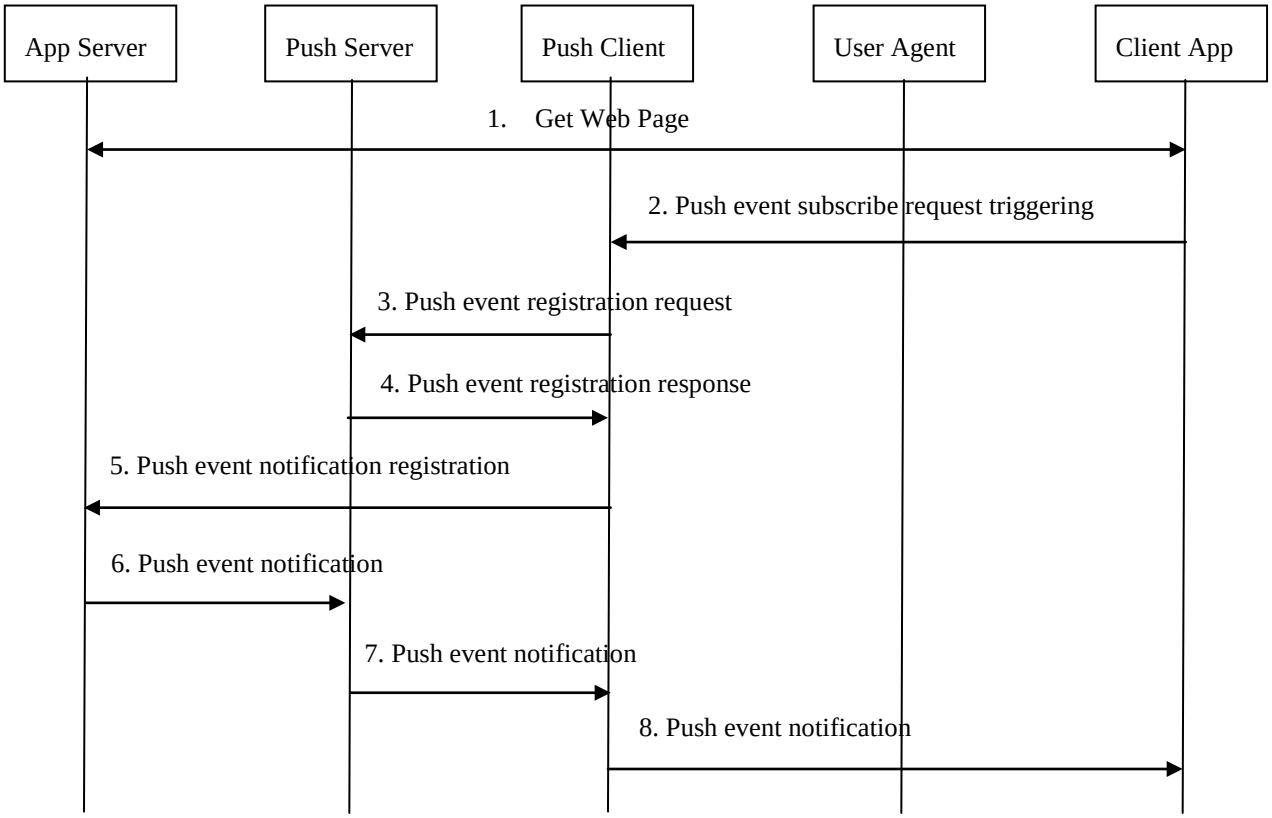

Figure 7: Push API Usage

Step 1: Web Application gets the web page from App Server. And it allows user to subscribe the push event.

Step 2: Push Client is triggered to get the web site identifier, and web site URL, from web application.

Step 3: Push Client sends registration request to Push Server. The registration request includes web site identifier and device identifier.

Step 4: Push Server receives the registration request, and generates receiver identifier, and stores the web site identifier, device identifier and receiver identifier. Then the Push Client receives registration response from Push Server, including receiver identifier.

Note that the main intention of identifier conversion is to protect user's privacy. This is implementation functionality for the Push Server, and it is out of scope of WRAPI 1.0 enabler.

Step 5: Push Client sends receiver identifier to web site, to register notification messages.

Step 6: If there are notification messages, App Server sends notification messages to Push Server, including receiver identifier.

Step 7: Push Server sends notification to push client on the user's device.

Step 8: Push Client sends push event notification to the Web Application.