# Simple XML over HTTP protocol for the Energy Interoperation OpenADR Profile

## Description

This document describes an application protocol for implementing the OpenADR services.

## References

* (1) Energy Interoperation Version 1.0 Working Draft 26
* (2) RFC-2616 HTTP/1.1 Status Code Definitions   
  <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
* (3) RFC-2616 HTTP/1.1 Header Field Definitions  
  <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>
* (4) RFC-2617 HTTP Authentication: Basic and Digest Access Authentication  
  <http://www.ietf.org/rfc/rfc2617.txt>
* (5) RFC-2246 Transport Layer Security Version 1.0  
  <http://tools.ietf.org/html/rfc2246>

## VTN/VEN Identifier

There are two options for identifying the VTN or VEN that is initiating a request:

* Use the authentication credentials as the identifier
* Include the identifier in the message

## Service Endpoint URIs

The endpoint names will be of the form:

https://<hostname>(:port)/(prefix/)OpenADR2/Simple/<service>

* “prefix” is an optional URI path prefix that may be used if a host is providing multiple
* “Simple” indicates the simple XML over HTTP protocol (others might be include “SOAP”, “BACnet”, etc.).
* <service> is the name of the EI service (e.g., “EiEvent”, “EiFeedback”, etc.).
* The “operation” portion of a service is defined by the XML payload sent in a request. E.g. a <eiRequestEventPayload /> root element specifies the EiRequestEvent operation.

## HTTP Methods

All messages will be sent using the HTTP POST method. This avoids caching and allows for an application level acknowledgement for each operation.

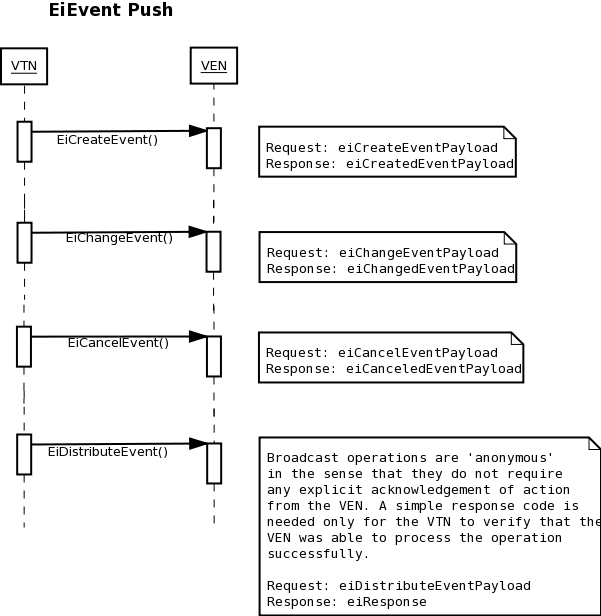
### Idempotence

HTTP transport layer safety, all operations are assumed to be idempotent (i.e. they can be executed any number of times without adverse side effects).

## Message Exchange Patterns

There are two MEPs used in OpenADR2 over HTTP: Push and Pull.

### Push

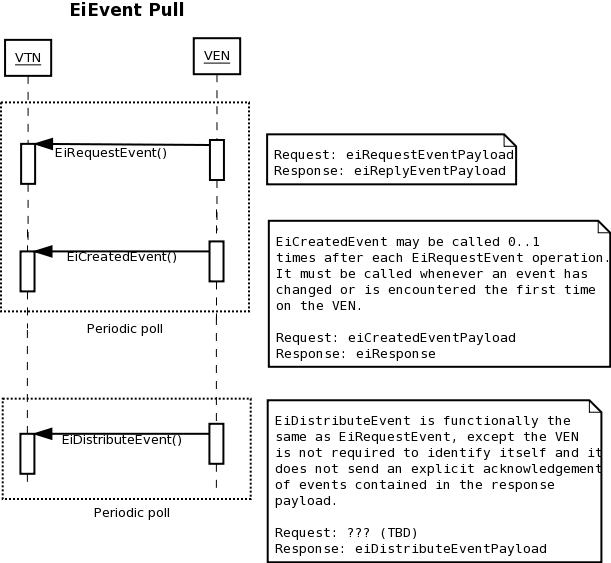
In a push MEP, messages may be sent from the VTN to VEN (pushed.) In order to use push, the VEN must expose HTTP URI endpoints (an HTTP server) to which the VTN may send requests such as eiCreateEvent. While this is the most efficient way to execute OpenADR2 over HTTP, it presents technical difficulties as the VEN may reside behind a network firewall.

Note that a ‘push’ VEN may still use the EiRequestEvent pull operation (detailed below) to request the current list of events. This might occur e.g. if the VEN reboots or otherwise loses state and needs to recover. In this case, the VEN would probably re-send an eiCreatedEvent response to re-acknowledge the list of returned events, which is acceptable since the eiCreatedEvent operation is idempotent.

### Pull

Using the pull MEP, all operations are initiated by the VEN to the VTN. This can be thought of as a ‘polling’ mode, where the VEN periodically asks for updates from the VTN. The pull mode removes the requirement for an HTTP server on the VEN, avoiding the technical limitation presented by the possibility of a network firewall in front of a VEN. However, the pull MEP has its own limitations, namely latency (due to limited polling frequency) and increased bandwidth requirements.

The pull MEP may involve a ‘two-phase’ execution to complete some operations. This is due to the nature of the VEN initiating the HTTP request. For example: in a push model, the VTN may notify a VEN of a new event via the EiCreateEvent operation. The VTN would send a request with an <eiCreateEventPayload/>, to which the VEN would respond with an <eiCreatedEventPayload/>.

However in the pull model, the VEN requests events from the VTN using EiRequestEvent, to which an <eiSentEventPayload/> is sent in the response. After parsing the response, the VEN still needs to acknowledge the creation of any new events by making a *second* request using the EiCreatedEvent operation on the VEN.

## Failure Conditions

The following failures can occur for a given operation:

* TCP (or below) fails
* HTTP fails (http error code)
* application acknowledgement fails (application error code)
* response failure (timeout or application error code)

The proper action for each failure condition depends upon the application and the operation being attempted. Since all operations are idempotent, it is safe to retry any operation.

## HTTP Response Codes

See (2) for more details on HTTP status code definitions.

**200 OK** - any response that the endpoint was able to handle completely and send a valid OpenADR response payload. This includes responses that may indicate an error at the application level (e.g. ‘you gave me an invalid event ID.’) Errors that indicate a failure at the transport level are handled by transport- level HTTP error codes:

**401 Unauthorized** - the requestor is not authorized to perform the given operation, likely due to missing or mis-configured authorization credentials. The requestor SHOULD NOT re-send the request until credentials have been modified.

**404 Not Found** - the VEN or VTN does not support the requested operation. The requestor SHOULD NOT re-send the request.

**406 Not Acceptable** – If a payload is sent that does not validate against the EI schema, or if a request content-type is unsupported. The requestor SHOULD NOT re-send the request without first modifying it.

**501 Not Implemented** – if any request is made with an unsupported HTTP method. The requestor SHOULD NOT re-send the request without fixing the HTTP method.

**503 Service Unavailable** – indicates that the server is temporarily unavailable, possibly due to inability to handle the current request load. This error in particular should indicate to the requestor that it should execute quiesce logic in order to not put further strain on the server. The requestor SHOULD retry the request after the proper quiesce period.

**500 Internal Server Error** – undefined or unexpected server error. The requestor MAY retry the request after a quiesce period.

For all error (non-200) codes, the content body of the response is undefined. The server MAY choose to send some informational message in the response, but the requestor is NOT obligated to parse or understand it.

All application-level error conditions are conveyed through the status code element of EiResponseType.

## Message Timeouts

Due to the near real-time nature of the communications, control of the message timeouts by the client in each interaction is required. Default HTTP timeouts can be as long as many minutes but periodic OpenADR operations can have a period of as small as a few seconds.

The timeout for client messages should be some fraction of the period of message transmission. *Needs more definition.*

There are two different types of timeout in HTTP: a connect timeout, or amount of time a requestor waits before the recipient accepts the TCP request, and a response timeout, or the amount of time the requestor waits for the responder to send the entire response.

## Message Retry/ Quiesce Behavior

When a request fails for any reason (either due to physical or network-level failure or a timeout) the requestor should institute ‘back-off’ or quiesce logic to avoid flooding the network or receiver with requests.

Clients should begin quiesce at some small interval (say, 1 second) plus or minus some random ‘jitter,’ which is a small percentage of that interval (say, 10%). So for example, the first quiesce interval for a device might be between 0.9 and 1.1 seconds. TODO define minimum back-off. Then the device may retry the request. If subsequent retries fail, quiesce interval should double from the prior interval, again adding a random jitter of plus or minus 10% of that interval. This doubling behavior for subsequent failures should continue up to some maximum, probably dependent on the poll interval in the case of a VEN polling a VTN. This is known as a “truncated binary exponential backoff algorithm.”

## Pull Timing

For VEN to VTN pull scenario, the following VEN parameters will be needed:

* configurable period (millisecond resolution)
* random start time within period (would be nice if the server knew the period)

TODO specify a minimum allowed poll period?

## HTTP Headers

The following HTTP headers (3) should appear in requests or responses (where indicated):

### Accept

The accept request header specifies the expected content-type of a response. Since responses are always “application/xml”, the Accept header MAY be omitted. However, if it is included, the value of the Accept request header should always be “application/xml”.

### Accept-Encoding

This request header indicates if a client supports content compression of the response payload. A VEN MAY include this header in a request if it supports content compression such as gzip or deflate. If the VEN includes this header, the VTN SHOULD honor it and compress the response content using one of the methods given in the request header.

### Authorization

The authorization header SHOULD be used where required to transmit necessary credentials to identify the requestor. See the “Security and Authentication” section below for more details.

TODO this presumes authorization and authentication are not baked into EI

### Content-Encoding

If a VTN is responding to a request for which it has compressed payload, it MUST include a Content-Encoding response header indicating the correct encoding method, such as gzip or deflate.

Push operations from a VTN SHOULD NOT utilize the content-encoding header in the request, since it would require the VTN to have *a priori* knowledge of which content-encodings are supported by each and every VEN.

### Content-Length

The content-length header MUST be used according to (3) to indicate content body size of all request and response payloads.

**Special Note:** ‘chunked’ transfer encoding (where content-length is unknown) is **not** a requirement for OpenADR 2.0, although it may be supported inherently by many HTTP/1.1 implementations. Implementers should assume that the total content body length is known when the response headers are sent, and should not attempt to send chunked responses.

### Content-Type

Should be used for both request and response messages, indicating payload MIME type. The appropriate value is “application/xml”. The content-type MAY also specify a character encoding. For OpenADR2, the only supported character encoding is UTF-8. If a charset is included, the entire header value should appear as “application/xml; charset=utf-8”.

### Host

The ‘Host’ header must be included in all requests per HTTP/1.1 Section 14.23 (3).

### User-Agent

The User-Agent header MAY be included by the requestor but its presence should not be relied upon, nor should it materially affect the behavior of the server handling the request.

### WWW-Authenticate

When a server responds with a 401 status, it MUST include this header to indicate proper authentication mechanisms. The requestor may then re-send a modified request, with proper Authorization header(s) included.

## Security and Authentication

TLS 1.0 (5) should be used to encrypt all traffic, regardless of the authorization method used. The client should always validate the server’s SSL certificate given during the handshake.

There are several common authentication mechanisms supported in HTTP. These are outlined as options for authentication in OpenADR 2.0 over HTTP. TODO need review by Security committee.

### SSL Client Certificate

In this scheme, the entity initiating the request must have a client X.509 certificate that is validated by the server during SSL/TLS negotiation. The server then verifies the identity of the client using the Common Name (CN) field of the certificate, and validates the certificate or a root certificate in the chain against the server’s trusted certificate store. This negotiation and validation occurs before any portion of the HTTP request is sent, meaning if negotiation fails, no HTTP response (or response code) is sent. Instead there is an error at the SSL protocol level.

### HTTP Basic or Digest Authorization

Authentication in HTTP may be done according to (4) which involves transmitting a username and password (either plaintext or hashed) in the HTTP Authorization header.

## Revision History

0.1, 5-Aug-2011, Dan Hennage  
0.2, 20-Sept-2011, Thom Nichols  
0.3, 5-Oct-2011, Thom Nichols  
0.4 11-Oct-2011, Thom Nichols