OpenADR 2.0
Profile Specification
B Profile

Updated 11-17-2015
Revision Number: 1.1
Document Status: **Final Specification**
Document Number: 20120912-1

Contact:	Editors:	Technical Director OpenADR Alliance:
OpenADR Alliance 275 Tennant Avenue, Suite 202 Morgan Hill, CA 95037 help@openadr.org	Ulrich Herberg, Fujitsu Jim Zuber, QualityLogic <JimZuber@qualitylogic.com> Daisuke Mashima, Fujitsu	Rolf Bienert <rolf@openadr.org>

Please send general questions and comments about the specification to comments@openadr.org

CONTENTS

OPEN AUTOMATED DEMAND RESPONSE

OpenADR 2.0 Profile Specification

FOREWORD

The development of the **Open Automated Demand Response Communications Specification**, also called OpenADR, began in 2002 following the California electricity crisis. The California Energy Commission Public Interest Energy Research Program funded an OpenADR research program through the Demand Response Research Center (DRRC) at Lawrence Berkeley National Laboratory (LBNL). OpenADR development began in 2002 to support California's energy policy objectives to move toward dynamic pricing to improve the economics and reliability of the electric grid. Initial field tests focused on automating a number of event-based DR utility programs for commercial and industrial (C&I) customers. The DRCC research set out to determine if today's communications and information technologies could be used to automate Demand Response (DR) operations using standardized electricity price and reliability signals. This research, development, and deployment have led to commercial adoption of OpenADR. Today, utilities and governments worldwide are using OpenADR to manage the growing demand for electricity and peak capacity of the electric systems. This low cost communications infrastructure is used to improve the reliability, repeatability, robustness, and cost-effectiveness of DR.

OpenADR is a fundamental element of U.S. Smart Grid interoperability standards being developed to improve optimization between electric supply and demand. OpenADR is designed to facilitate automated DR actions at the customer location, whether it involves electric load shedding or shifting. OpenADR is also designed to provide continuous dynamic price signals such as hourly day-ahead or day-of real time pricing. OpenADR has been field tested and deployed in a number of DR programs in U.S and worldwide. While the scope of OpenADR focuses on signals for DR events and prices, significant work focuses on DR strategies and techniques to automate DR within facilities. OpenADR interacts with facility control systems that are pre-programmed to take action based on a DR signal, enabling a response to a DR event or a price to be fully automated, with no manual intervention.

The DRCC OpenADR 1.0 specification was donated to the Organization of Structured Information Standards (OASIS) to create a national standard for OpenADR. The OASIS' Energy Interoperation (EI) Technical Committee (TC) developed a standard to describe "an information model and a communication model to enable collaborative and transactive use of energy, service definitions consistent with the OASIS SOA Reference Model [SOA-RM], and XML vocabularies for the interoperable and standard exchange of dynamic price signals, reliability signals, emergency signals, communication of market participation information such as bids, load predictability and generation information." Considering that the goal of OASIS EI TC was more than DR and Distributed Energy Resources (DER), the EI TC created profiles within the EI Version 1.0 standard for specific applications within the Smart Grid. The OpenADR Alliance used the EI OpenADR profile as the basis for the OpenADR 2.0 Profile Specification defined in this document. OpenADR 2.0 defines profiles for DR and Distributed Energy Resources (DER), while keeping in mind the requirements of the diverse market and stakeholder needs.

INTRODUCTION

Development of the Demand Response (DR) market has resulted in a transition from manual DR to OpenADR in Automated DR (Auto-DR) programs. As of 2013, over 250 MW was enrolled in California commercial and industrial customers Auto-DR programs using OpenADR 1.0.[1] DR is defined as "…action taken to reduce electricity demand in response to price, monetary incentives, or utility directives so as to maintain reliable electric service or avoid high electricity prices."[2] OpenADR 1.0 was developed to support Auto-DR programs and California's energy policy objectives to move toward dynamic pricing to improve the economics and reliability of the electric grid. The recent developments have expanded the use of OpenADR to meet diverse market needs such as ancillary services (Fast DR), dynamic prices, intermittent renewable resources, supplement grid-scale storage, electric vehicles, and load as generation. For example, with real-time price information, an automated client within the customer facility can be designed to continuously monitor these prices and translate this information into continuous automated control and response strategies. This rationale is a fundamental element of the United States (U.S.) Smart Grid interoperability standards, which are developed to improve dynamic optimization of electric supply and demand.

OpenADR Communications have the following defining features:

- **Continuous, Secure, and Reliable** - Provides continuous, secure, and reliable two-way communications infrastructures where the end points at the end-use site receive and acknowledge the receipt of DR signals from the energy service providers.
- **Translation** - Translates DR event information to continuous Internet signals to facilitate DR automation. These signals are designed to interoperate with energy management and control systems, lighting, or other end-use controls.
- **Automation** - Receipt of the external signal is designed to initiate automation through the use of pre-programmed Demand Response strategies determined and controlled by the end-use participant.
- **Opt-Out** - Provides opt-out or override function to any participants for a DR event if the event comes at a time when changes in end-use services are not desirable.
- **Complete Data Model** - Describes a rich data model and architecture to communicate price, reliability, and other DR activation signals.
- **Scalable Architecture** - Provides scalable communications architecture to different forms of DR programs, end-use buildings, and dynamic pricing.
- **Open Standards** - Open standards-based technology such as Internet Protocol (IP) and web services form the basis of the communications model.

OpenADR is a communications data model, along with transport and security mechanisms, which facilitate information exchange between two end-points, the electricity service provider and the customer. It is not a protocol that specifies "bit-structures" as some communications protocols do, but instead relies upon existing open standards such as eXtensible Mark-up Language (XML)

[1] Piette, Mary Ann, Girish Ghatikar, Sila Kiliccote, Ed Koch, Dan Hennage, Peter Palensky, and Charles McParland. 2009. Open Automated Demand Response Communications Specification (Version 1.0). California Energy Commission, PIER Program. CEC□500□2009□063.

[2] U.S. Federal Energy Regulatory Commission (FERC), 2007 Assessment of Demand Response and Advanced Metering, Staff Report, available: http://www.ferc.gov/legal/staff-reports/09-07-demand-response.pdf

and Internet Protocol (IP) as the framework for exchanging DR signals. In some references the term “system,” “technology,” or “service” is used to refer to the features of OpenADR.

OpenADR is designed to facilitate automation of DR actions at the customer location, whether it involves electric load shedding or load shifting. We are often asked if the communications data model can be used for continuous operations. The answer is **yes**. Many emergency or reliability DR events occur at specific times when the electric grid is strained. The OpenADR communications are designed to coordinate such signals with facility control systems (commercial, industrial, and residential). OpenADR is also designed to provide continuous dynamic price signals such as hourly day-ahead or day-of real time pricing. With such price information an automated client can be configured to continuously monitor these prices and translate this information into continuous automated control and response strategies within a facility. Several reports present the history of OpenADR 1.0 research.[3] This OpenADR 2.0 profile specification covers the signaling data models for price and reliability signals to both wholesale and retail markets in the U.S.

OpenADR provides the following benefits:

- **Open Specification**–Provides a standardized DR communications and signaling infrastructure using open, non-proprietary, industry-approved data models that can be implemented for both dynamic prices and DR emergency or reliability events.
- **Flexibility**–Provides open communications interfaces and protocols that are flexible, platform-independent, interoperable, and transparent to end-to-end technologies and software systems.
- **Innovation and Interoperability**–Encourages open innovation and interoperability, and allows controls and communications within a facility or enterprise to build on existing strategies to reduce technology operation and maintenance costs, stranded assets, and obsolesce in technology.
- **Ease of Integration**–Facilitates integration of common Energy Management and Control Systems (EMCS), centralized lighting, and other end-use devices that can receive Internet signals (such as XML).
- **Supports Wide Range of Information Complexity –** Can express the information in the DR signals in a variety of ways to allows for systems ranging from simple end devices (e.g., thermostats) to sophisticated intermediaries (e.g., aggregators) to receive the DR information that is best suited for its operations.
- **Remote Access**– Facilitates opt-out or override functions for participants to manage standardized DR-related operation modes to DR strategies and control systems.

The OpenADR Alliance is the primary authority for the development and adoption of OpenADR, leveraging the OpenADR 1.0 activities and OASIS Energy Interoperation (EI) Technical Commit-

[3] These reports are available at http://drrc.lbl.gov/drrc-pubsall.html:

Piette, M.A., S. Kiliccote, G. Ghatikar, Design and Implementation of an Open, Interoperable Automated Demand Response Infrastructure, Proceedings of the Grid-Interop Forum, October 2007, LBNL-63665.

Koch, E., M.A. Piette, Architecture Concepts and Technical Issues for an Open, Interoperable Automated Demand Response Infrastructure. Proceedings of the Grid-Interop Forum, October 2007. LBNL-63664.

Piette, M.A, D. Watson, N. Motegi, S. Kiliccote Automated Critical Peak Pricing Field Tests: 2006 Pilot Program Description and Results, August, 2007. LBNL-62218.

Motegi, N., M.A. Piette, D.S. Watson, S. Kiliccote, P. Xu. Introduction to Commercial Building Control Strategies and Techniques for Demand Response, May 2007. LBNL-59975.

tee's Version 1.0 standard.[4] The OpenADR profile within OASIS EI Version 1.0 standard is the basis for the OpenADR 2.0 profile specification and is referenced as appropriate in this document.

[4] Energy Interoperation OASIS Committee Specification, Energy Interoperation Version 1.0, December 2011. http://www.oasis-open.org/committees/download.php/44364/energyinterop-v1.0-csprd03.zip

1 Scope

The OpenADR 2.0 profile specification is a flexible data model to facilitate common information exchange between electricity service providers, aggregators, and end users. The concept of an open specification is intended to allow anyone to implement the two-way signaling systems, providing the servers, which publish information (Virtual Top Nodes or VTNs) to the automated clients, which subscribe the information (Virtual End Nodes, or VENs).

This OpenADR 2.0 profile specification covers the signaling data models between VTN and VEN (or VTN/VEN pairs) and does include information related to specific DR electric reduction or shifting strategies, which are taken at the facility. In particular, OpenADR 2.0 supports the following services from OASIS EI Version 1.0 standard or subset thereof. Extensions to these services are included to meet the DR stakeholder and market requirements:

1. **Registration (EiRegisterParty)**: Register is used to identify entities such as VEN's and parties. This is necessary in advance of an actor interacting with other parties in various roles such as VEN, VTN, tenderer, and so forth.
2. **Enrollment (EiEnroll)**: Used to enroll a Resource for participation in DR programs. This establishes a relationship between two actors as a basis for further interactions. (Planned for future releases)
3. **Market Contexts (EiMarketContext)**: Used to discover program rules, standard reports, etc. Market contexts are used to express market information that rarely changes, and thereafter need not be communicated with each message. (Planned for future releases)
4. **Event (EiEvent)**: The core DR event functions and information models for price-responsive DR. This service is used to call for performance under a transaction. The service parameters and event information distinguish different types of events. Event types include reliability events, emergency events, and more – and events MAY be defined for other actions under a transaction.
5. **Quote or Dynamic Prices (EiQuote)**: EiDistributeQuote for distributing complex dynamic prices such as block and tier tariff communication. These are sometimes referred to as *price signals*; such signals are indications of a possible tender price – they are not themselves actionable. Such services can be used to implement the functionality for energy market interactions or transactional energy. (Planned for future releases)
6. **Reporting or Feedback (EiReport)**: The ability to set periodic or one-time information on the state of a Resource (response).
7. **Availability (EiAvail)**: Constraints on the availability of Resources. This information is set by the end node and indicates when an event may or may not be accepted and executed by the VEN with respect to a Market Context. Knowing the Availability and Opt information for its VENs improves the ability of the VTN to estimate response to an event or request. (Planned for future releases)
8. **Opt or Override (EiOpt)**: Overrides the EiAvail; addresses short-term changes in availability to create and communicate Opt-in and Opt-out schedules from the VEN to the VTN.

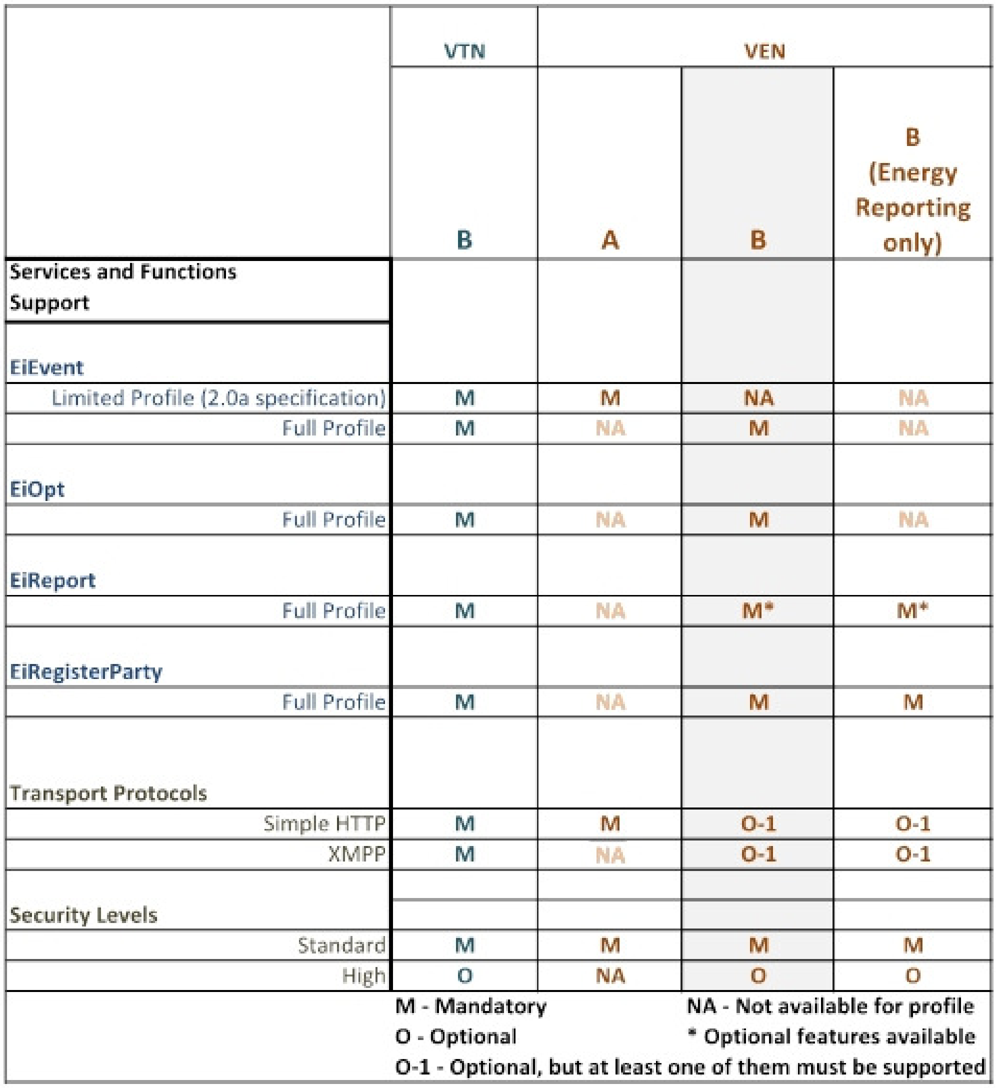
These OpenADR 2.0 services in this specification provide information that is pertinent to DR, pricing, and DER communication requirements. These services make no assumption on specific DR electric load control strategies within the resource or market-specific contractual or business agreements between electricity service providers and their customers.

OpenADR uses an application-level data model, which is independent of transport mechanisms. For the purposes of interoperability, OpenADR 2.0 provides basic transport mechanisms and their relevant interaction patterns (e.g., PUSH information vs. PULL information) to address different stakeholder needs.

OpenADR 2.0 specifies the necessary level of security that is essential to meet the U.S. Cyber Security requirements for such purposes as data confidentiality, integrity, authentication and message-level security. Such security requirements are essential for non-repudiation and to mitigate any resulting Cyber Security risks.

OpenADR 2.0 provides a clear set of mandatory and optional attributes within each of the services to meet the broader interoperability, testing and certification requirements, while creating feature-sets with different product profiles to address today's market needs as well as future requirements that are closely aligned to meet OpenADR goals and national interoperability requirements for Smart Grid standards.

The different product certification levels for OpenADR include OpenADR 2.0a, OpenADR 2.0b, and OpenADR 2.0b "Energy Reporting only" VENs (depicted in Figure 1). VTN certification for 2.0a will end with publication of this document, and existing implementations of 2.0a VTNs must upgrade to the OpenADR2.0b standard. For this reason, Figure 1 has no column for 2.0a VTN. 2.0b VTNs must support 2.0a VENs (and therefore comply with the OpenADR2.0a standard). VENs can be certified using the 2.0a, the 2.0b, and a 2.0b "Energy reporting only" profile. An OpenADR 2.0c or new market-specific profiles may be specified in the future. This profile specification describes OpenADR 2.0b. For the final 2.0a features, please refer to the respective specification, which is available on the OpenADR Alliance's website – http://www.openadr.org/.



	VTN	VEN		
	B	A	B	B (Energy Reporting only)
Services and Functions Support				
EiEvent				
Limited Profile (2.0a specification)	M	M	NA	NA
Full Profile	M	NA	M	NA
EiOpt				
Full Profile	M	NA	M	NA
EiReport				
Full Profile	M	NA	M*	M*
EiRegisterParty				
Full Profile	M	NA	M	M
Transport Protocols				
Simple HTTP	M	M	O-1	O-1
XMPP	M	NA	O-1	O-1
Security Levels				
Standard	M	M	M	M
High	O	NA	O	O

M - Mandatory NA - Not available for profile
O - Optional * Optional features available
O-1 - Optional, but at least one of them must be supported

Figure 1 OpenADR 2.0 Certification Levels

2 Normative References

- **[OpenADR 2.0 PICS]** – Source: OpenADR website, http://www.openadr.org
- **[OpenADR 2.0 Certificate Policy]** – Source: OpenADR website, http://www.openadr.org
- **[OASIS EI 1.0]** Energy Interoperation OASIS Committee Specification 02, *Energy Interoperation Version 1.0*, http://docs.oasis-open.org/energyinterop/ei/v1.0/cs02/energyinterop-v1.0-cs02.html, February 2012.
- **[OASIS EMIX 1.0]** EMIX OASIS Committee Specification 02, *Energy Market Information Exchange 1.0*, http://docs.oasis-open.org/emix/emix/v1.0/cs02/emix-v1.0-cs02.html, January 2012.
- **[OASIS WS-Calendar]** WS-Calendar OASIS Committee Specification 1.0, *WS-Calendar*, http://docs.oasis-open.org/ws-calendar/ws-calendar-spec/v1.0/cs01/ws-calendar-spec-v1.0-cs01.html, July 2011.
- **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
- **[RFC2246]** T. Dierks, C. Allen, *The TLS Protocol Version 1.0*, http://www.ietf.org/rfc/rfc2246.txt, IETF RFC 2246, January 1999.
- **[RFC2616]** R. Fielding et. al., *Hypertext Transfer Protocol -- HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, IETF RFC 2616, June 1999.
- **[RFC3275]** D. Eastlake, J. Reagle, D. Solo, *(Extensible Markup Language) XML-Signature Syntax and Processing*, http://www.ietf.org/rfc/rfc3275.txt, IETF RFC 3275, March 2002.
- **[RFC3986]** T. Berners-Lee et. al., *Uniform Resource Identifier (URI): Generic Syntax*, http://www.ietf.org/rfc/rfc3986.txt, IETF RFC 3986, June 1999.
- **[RFC4346]** T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.1*, http://www.ietf.org/rfc/rfc4346.txt, IETF RFC 4346, April 2006.
- **[RFC5246]** T. Dierks, E. Rescorla, The *Transport Layer Security (TLS) Protocol Version 1.2*, http://www.ietf.org/rfc/rfc5246.txt, IETF RFC 5246, April 2008.
- **[RFC6120]** P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core Version 1.0*, http://www.ietf.org/rfc/rfc6120.txt, IETF RFC 6120, March 2011.
- **[RFC6121]** P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*, http://www.ietf.org/rfc/rfc6121.txt, IETF RFC 6121, March 2011.
- **[RFC6122]** P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Address Format*, http://www.ietf.org/rfc/rfc6122.txt, IETF RFC 6122, March 2011.
- **[SOA-RM]** SOA-RM OASIS Standard, *OASIS Reference Model for Service Oriented Architecture 1.0*, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html, October 2006.

- **[XEP-0030]** Joe Hildebrand et. al., *XEP-0030: Service Discovery*, http://xmpp.org/extensions/xep-0030.html, June 2006.

3 Non-Normative References

- OpenADR 2.0a Profile Specification
- UCA OpenSG OpenADR security profile
- IRC and NAESB requirements/use-cases
- NIST Special Publication 800-131A
- NAESB REQ.21 Energy Services Provider Interface (ESPI), Version 1.0 October 2011(Green Button)

4 Terms and Definitions

The OpenADR Alliance: The OpenADR Alliance is comprised of industry stakeholders that are interested in fostering the deployment of low-cost price- and reliability-based Demand Response communication protocol by facilitating and accelerating the development and adoption of OpenADR standards and compliance with those standards. These include de facto standards based on specifications published by LBNL in April 2009, as well as Smart Grid-related standards emerging from OASIS, UCAIug, NAESB, and IRC.

OpenADR 2.0 Profile Specification: The OpenADR 2.0a and 2.0b Profile Specifications provide specific implementation related information in order to build an OpenADR enabled device or system. Developers shall use the Profile Specification in conjunction with the schemas, sample payloads, PICS and test plans.

OASIS Energy Interoperation (EI): Energy Interoperation standard describes information and communication model to coordinate energy supply, transmission, distribution, and use, including power and ancillary services, between any two parties, such as energy suppliers and customers, markets and service providers, in any of the domains defined in the Smart Grid. The EI 1.0 standard was used as a basis for OpenADR 2.0 Profile Specification.

Demand Response: A mechanism to manage customer load demand in response to supply conditions, such as prices or availability signals.

Slow DR: Demand Response where the signals are sent significantly before the events are called, such as day-ahead.

Fast DR: Fast Demand Response or Fast DR refers to programs that require a (much) faster than usual response time. While typical peak shaving DR programs have minutes, if not hours or days, of lead-time, these programs have lead times of seconds (e.g., 4 second response time) used for load balancing and frequency stabilization (e.g., ancillary services and regulation services)

PUSH/PULL operations: OpenADR 2.0 can be used in either PULL mode (VEN pulling information from VTN) or in a PUSH mode in the simple HTTP transport. The XMPP transport uses a PUSH model, although VENs can still make requests of the VEN, excluding the use of oadrPoll.

Simple HTTP: Simple HTTP in OpenADR 2.0 (a/b) refers to an HTTP implementation that uses HTTP POST over TLS to propagate OpenADR payloads.

The upper-case 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 [RFC2119].

5 Abbreviations

AutoDR: Automated Demand Response

DER: Distributed Energy Resources

DR: Demand Response

DRRC: Demand Response Research Center

DUT: Device Under Test

EI: Energy Interoperation

HTTP: Hyper Text Transfer Protocol

IRC: ISO/RTO Council

ISO: Independent Systems Operator

JID: Jabber Identifiers

LBNL: Lawrence Berkeley National Laboratory

NAESB: North American Energy Standards Board

OASIS: Organization of Structured Information Standards

OpenADR: Open Automated Demand Response

PICS: Protocol Implementation Conformance Statement

RTO: Regional Transmission Operators

SASL: Simple Authentication and Security Layer

Simple HTTP: Limited REST transport protocol

SOAP: Simple Object Access Protocol

TC: Technical Committee

UCAIug: Utilities Communications Architecture International Users Group

VEN: Virtual End Node

VTN: Virtual Top Node

XMPP: XML Messaging and Presence Protocol

6 Overview

This section gives an overview of the message exchanges, the roles, and actors supported within OpenADR 2.0 Profile Specifications. It contains the following elements, used to develop test and certification framework for Smart Grid and customer systems interoperability:

1. A set of data models derived from the OASIS Energy Interoperation 1.0 standard.
2. A set of services for performing various functions and operations for the exchange of the data models, also derived from the OASIS Energy Interoperation 1.0 standard.
3. A set transport mechanisms for implementing the services. The transport mechanisms rely upon standard-based IP communications such as HTTP and XML Messaging and Presence Protocol (XMPP).
4. A set of security mechanisms for securing each of the transport mechanisms.
5. OpenADR 2.0 Schemas (separate document)

The OASIS Energy Interoperation (EI) 1.0 standard describes the coordination of energy supplies, transmission, distribution, and usage. However, as shown in Figure 2, the features required within the OpenADR 2.0 Profile Specifications are a subset of this Energy Interoperation 1.0 standard. The reason for this development is twofold – 1) By having a strict and clear subset of features to support OpenADR, devices can be clear about what features they must support in order to participate in the OpenADR ecosystem; 2) by being a referenced subset, devices can validate against the Energy Interoperation 1.0 standard, and participate in that ecosystem as well. These requirements are critical to maintain interoperability and reference the original standard.

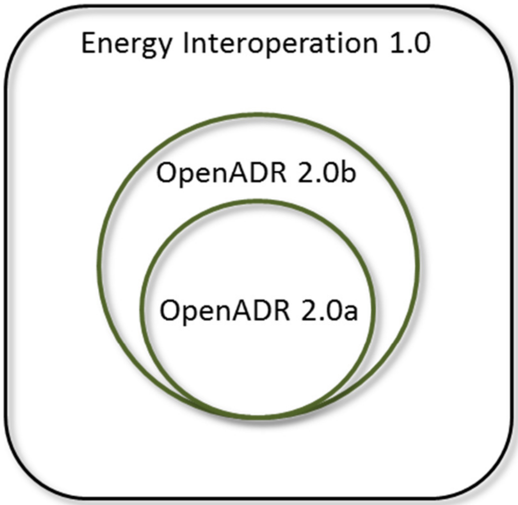


Figure 2 Relationship between OASIS Energy Interoperation 1.0 Standard and OpenADR 2.0 Profiles

The above Figure 2 shows the relative relationship between the complete data model and services features of the Energy Interoperability 1.0 standard and OpenADR 2.0 profiles. The diagram also shows how the different profile subsets of OpenADR 2.0 relate to the complete OpenADR 2.0 feature set.

Message exchanges in OpenADR 2.0 support services related to communicating information about Demand Response events. Networks of OpenADR nodes must be able to query for active or pending events, register themselves, schedule events, and send reports. OpenADR nodes must also be able to refine and update previously sent information. For instance, an OpenADR node reporting DR events to nodes downstream must be able to cancel a previously scheduled event if this becomes necessary.

Nodes in these networks are divided into two groups: nodes that publish and transmit information about events to other nodes (e.g., utilities), and nodes that receive the communications respond to that information (e.g., end users). The upstream nodes that publish information about upcoming events are called Virtual Top Nodes (VTNs); the downstream nodes that receive this information are called Virtual End Nodes (VENs).
These nodes may communicate using a variety of protocols. They may communicate using HTTP in either PUSH mode (where the VTN initiates communication) or in a PULL mode (the VEN requests information from the VTN to begin a series of message exchanges). The VTNs/VENs may also communicate over other transport mechanisms such as XML Messaging and Presence Protocol (XMPP).

The profiles support end devices with varying degree of capabilities. The profile levels are 2.0a, for simple devices, and 2.0b, for more sophisticated devices. An OpenADR 2.0c profile may be specified in the future. Profile 2.0b VEN devices support a sub-profile referred to as Report Only. Each profile has a defined set of optional and mandatory functionality. In general, this functionality is only optional for VEN, as VTN must support all functionality for a given profile in order to maintain interoperability.

6.1 Node and Device Types

Following the notion from the OASIS Energy Interoperability 1.0 standard, OpenADR 2.0 uses the definitions of Virtual Top Nodes (VTNs) and Virtual End Nodes (VENs). For any interaction between actors using OpenADR 2.0 to communicate, one actor is designated the Virtual Top Node and the remainder are the Virtual End Nodes. All communications are between a VTN and one or more VEN's. There is no peer-to-peer communication in OpenADR 2.0, i.e., VTNs do not communicate directly with other VTNs and likewise VENs do not communicate directly with other VENs.

Generally in an interaction, the VTN acts as the server, providing information to the VEN, which themselves respond to the information. For instance, a VTN would be the entity to announce a DR event; VENs hear about DR events and respond. The response may be to reduce power to some devices. The response could also be to propagate the signal further downstream to other VEN's. In this case, the VEN would become the VTN for the new interaction (as depicted in Figure 3).

For the purpose of device development, the OpenADR Alliance always tests the interface between a VTN and a VEN, where either node can be the Device Under Test (DUT). Intelligence build into the systems not related to the OpenADR 2.0 message exchange is not part of the OpenADR Alliance testing program.

The authoritative requirements for implementation of OpenADR VENs and VTNs are defined in the OpenADR schema and the conformance rules. Narrative descriptions in later sections separate from the conformance rules provide context and implementation examples but do not contain the full breath of implementation requirements. Therefore, we strongly recommend that implementers thoroughly understand ALL the conformance rules prior to coding.

Virtual Top Node (VTN): An entity that is responsible for communicating grid conditions (e.g., prices, reliability events, etc.) to other entities (i.e., VENs) that control demand side resources. The VTN is able to communicate with both the Grid and the VEN devices or systems in its domain. A VTN may take the role of a VEN interacting with another VTN.

Virtual End Node (VEN): The VEN has operational control of a set of resources and/or processes and is able to control the electrical energy demand of these in response to an understood set of Smart Grid messages (i.e., DR signals). The VEN may be either a producer or consumer of energy. The VEN is able to communicate (2-way) with a VTN receiving and transmitting Smart Grid messages that relay grid situations, conditions, or events. A VEN may take the role of a VTN in other interactions.

Although within any interaction, one actor is designated the VTN and the remainder are VENs (moreover, most interactions have exactly one VTN and one VEN), sets of actors can be arranged in any hierarchy, by allowing actors to act as VENs for some interactions and VTNs for others.

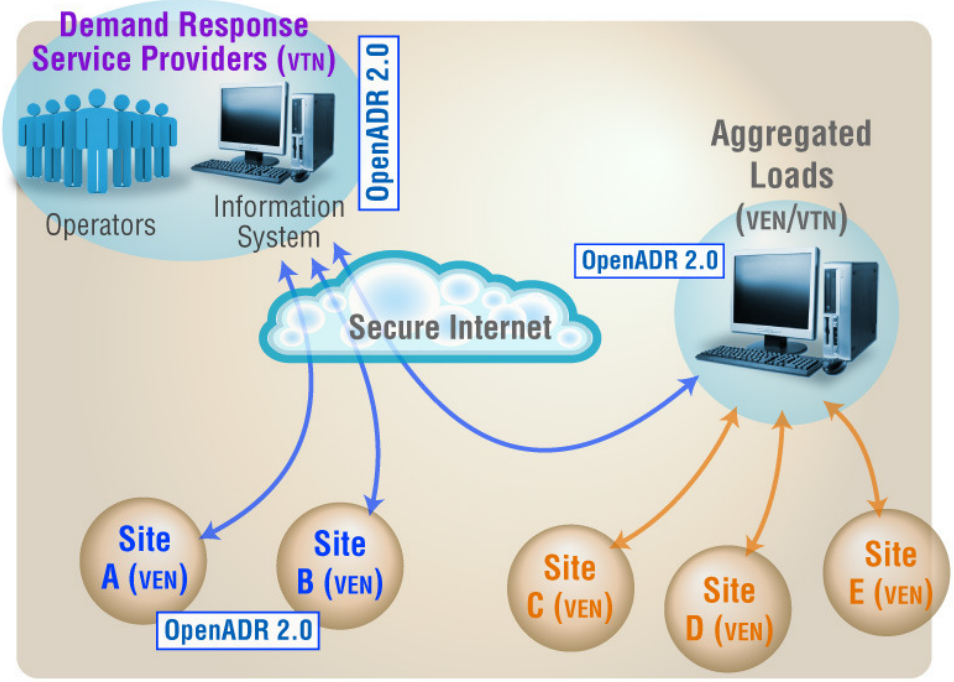


Figure 3 Possible relationships of VTN and VEN

As illustrated in Figure 3, any combination of VTN and VEN is possible through a utility/ISO (service provider or server) to sites (customers). Also, as shown above, systems can function as a VEN to a VTN in a higher layer of the hierarchy and as a VTN to subordinate VENs. In either of these architectural scenarios, an operation can be initiated by the VTN to a VEN (PUSH pattern) or a VEN can request it from a VTN (PULL pattern). The exchanged events in either direction can be independent from each other and the OpenADR Alliance does not define how the nodes react to the information. In nodes, which support both the VTN and VEN interfaces (e.g., aggregators) there are no specifications or constraints on how messages arriving at the VEN interface are coupled or translated into any subsequent messages that may be sent from the VTN interface and vice versa. They are treated as completely independent interfaces and both will be evaluated and tested independently to assure adherence to the profile specification and interoperability. A specific deployment scenario depends on an agreement between the utility/ISO and the participating sites.

6.2 Energy Interoperation Services

The OASIS Energy Interoperation specification encompasses a number of services for collaborative and transactive use of energy of which only 4 services are applicable for OpenADR (in the current B profile). OpenADR 2.0 uses a profiled subset of the Energy Interoperation services tailored to meet the OpenADR needs, but still conforming to the Energy Interoperation Specification.

These services are EiRegisterParty, EiEvent, EiReport, and EiOpt. For further information on these services, consult Section 7.

6.3 Feature Sets

The OpenADR 2.0 specification has been designed to support a variety of end-use devices, from simple to more sophisticated. Accordingly, not all devices need to support all OpenADR capabilities. For example, although OpenADR describes how VTNs may report lists of active and pending future events, a simple residential programmable thermostat may only need information about the next pending event. In order to prevent onerous requirements on all devices, OpenADR currently defines two feature sets, each of which represent a subset of OpenADR functionality. The feature sets are 2.0a (or simply A), and 2.0b (or simply B), and possibly a future specification of a 2.0c profile. The purpose of the profiles is to create a range of functionality that can be supported by devices as simple as a thermostat (profile A) to more complex IT based systems such as might be used by aggregators (profile B). Additional feature sets can be established to accommodate additional market requirements.

6.4 Assumptions

This is a list of operating assumptions regarding correct functional behavior between VENs and VTNs:

- Both the VTN and VEN have a reasonably accurate awareness of the current time. How that is accomplished is device and/or deployment specific.

7 OpenADR 2.0 Feature Set Profiles

The OpenADR Alliance has defined several feature sets to accommodate a variety of different devices for varying applications. Each Feature Set Profile describes the required services to be implemented as well as the accompanying operations and attributes. Please refer to the OpenADR 2.0 Protocol Implementation Conformance Statement ([OpenADR 2.0 PICS] – refer to separate document for alliance members only) for more information regarding the required services. Additional profiles will be added if required by the market participants.

The OpenADR 2.0 services discussed in the Feature Set Profiles can be found in section 8. All services are subsets of the OASIS Energy Interoperation Specification and validate individually with the relevant schemas.

This document outlines the OpenADR 2.0b profile. Any following profiles will have their own sets of documents and will follow. Summary information for these profiles is described in this document.

7.1 Differences between OpenADR 2.0a and OpenADR 2.0b

The only service supported by the A profile is the EiEvent service. The EiEvent object is simplified in the A profile with the following constraints:

- Only one signal per event is allowed and that signal must be the OpenADR well-known signal SIMPLE.
- There is a limited event targeting with only venID, groupID, resourceID, and partyID supported. (eiEvent:eiTarget).
- Targeting at the signal level with device classes is not supported (eiEventSignal:eiTarget:endDeviceAsset).
- Baselines are not supported (eiEvent:eiEventSignals:eiEventBaseline).
- modificationDateTime and modificationReason are not supported.
- The endpoint URL for simple HTTP in 2.0b is:
  https://<hostname>(:port)/(prefix/)OpenADR2/Simple/**2.0b**/<service>
  (note the additional “2.0b/” prefix compared to the 2.0a specification).

B profile VTNs must be able to interoperate with both A and B VENs. B VENs may optionally support the A profile, but are not required to. If a B profile VEN also supports the A profile and communicates with an A profile VTN, it must constrain its behavior as follows:

- Use only the EiEvent service
- Do not send oadrPoll
- Root Payload Element schemaVersion attribute must not be sent as part of any payload
- oadrResponse:venID must not be sent as part of the oadrResponse payload

When a B profile VTN communicates with an A profile VEN, it must constrain its behavior as follows:

- Use only the EiEvent service

- Events must contain no more than one EiEventSignal
- Event signalName must be “SIMPLE” as defined in the A profile conformance rule 7
- The event signalPayload values must limited to 0,1,2,3 as defined in A conformance rule 9
- The following optional B profile eiEvent schema elements must not be sent as part of the oadrDistributeEvent payload to the VEN:
    - eiEvent:eventDescriptor:modificationDateTime
    - eiEvent:eventDescriptor:modificationReason
    - eiEvent:eiTarget:aggregatedPnode
    - eiEvent:eiTarget:endDeviceAsset
    - eiEvent:eiTarget:meterAsset
    - eiEvent:eiTarget:pnode
    - eiEvent:eiTarget:serviceArea
    - eiEvent:eiTarget:serviceDeliveryPoint
    - eiEvent:eiTarget:serviceLocation
    - eiEvent:eiTarget:transport Interface
    - eiEvent:eiEventSignals:eiEventSignal:eiTarget
    - eiEvent:eiEventSignals:eiBaseline
    - eiEvent:eiEventSignals:eiEventSignal:itemBase Substitution Group
- Root Payload Element schemaVersion attribute must not be sent as part of any payload
- The currentValue eiEvent element must be included in the oadrDistributeEvent payload

The OpenADR 2.0b profile adds the functionality excluded above for the 2.0a profile.

In addition, the 2.0b profile optionally supports XML signatures, which requires a different XML root element for each payload (refer to Section 10.6). With the changes to the B schema to support XML signatures, the changes required for A and B devices to interoperate will be much more extensive and may not be practical using a common code base. The B devices would need to strip the root oadrPayload, Signature, and oadrSignedObject elements from every payload.

7.2 OpenADR 2.0b Feature Set Profile

The OpenADR 2.0b Feature Set was developed for advanced Demand Response systems and markets (e.g., wholesale and retail DR markets). It includes an extended EiEvent Service as well as several additional services usable in Demand Response programs and for ancillary services.

7.2.1 Supported Services

a) EiEvent Service

b) EiReport Service

c) EiRegisterParty Service

d) EiOpt Service

7.2.2 Report Only VENs

Some devices, such as meters, do not have the ability to shed load. However, these types of devices can provide valuable reporting information to the VTN. The B profile has a sub-profile for VENs called Report Only, which includes support for the following services:

a) EiReport Service

b) EiRegisterParty Service

7.2.3 Transport Mechanism

Supported transport mechanisms are as follows. The mechanisms are described in section 9.

a) HTTP is mandatory for VTNs; VENs must either support HTTP or XMPP

b) XMPP is mandatory for VTNs; VENs must either support HTTP or XMPP

7.2.4 Security

Supported security details are outlined in sections 9 and 10. The following security levels apply to OpenADR 2.0b.

a) Standard Security – mandatory

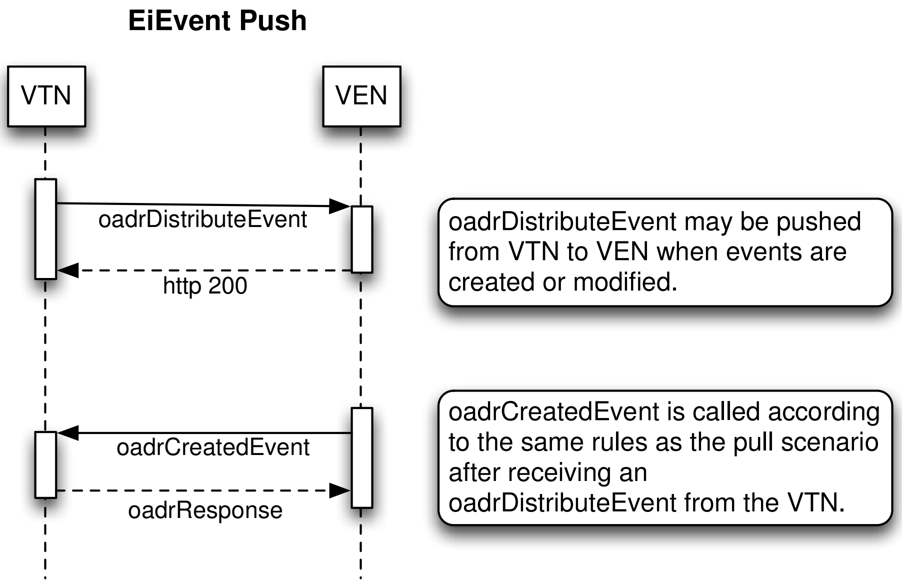
b) High Security – optional

8 OpenADR 2.0b Services and Data Models Extensions

8.1 OpenADR 2.0b EiEvent Service

Events are generated by the VTN and sent to the VEN using the oadrDistributeEvent payload containing one or more events described by the oadrEvent element. Some events require a response and others do not as indicated by the oadrResponseRequired element in the event description. If a response is required, the VEN acknowledges its opt-in or out-out disposition by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent. If no response is required, the VEN must not reply with oadrCreatedEvents (or oadrCreateOpt) payloads for this event.

Either a PUSH or PULL interaction pattern may be used. For push, the VTN will deliver events to the VEN using an oadrDistributeEvent payload. In PULL mode, the oadrDistributeEvent will be sent from the VTN to the VEN as response to an oadrPoll (refer to section 8.6). In addition to periodically sending oadrPoll, a VEN may also send one-time oadrRequestEvent payloads to the VTN in order to acquire events from the VTN. Note that 2.0A VENs will use oadrRequestEvent for periodic pulling of events instead of oadrPoll. If an application level response is required, the VEN asynchronously sends an oadrCreatedEvent back to the VTN in a second message. These sequences are illustrated in Figure 4. Note that in simpleHTTP PUSH mode, the VEN's response to oadrDistributeEvent is a transport level acknowledgement if required (in the case of HTTP a 200 response code, in XMPP an empty *iq* stanza).



oadrDistributeEvent may be pushed from VTN to VEN when events are created or modified.

oadrCreatedEvent is called according to the same rules as the pull scenario after receiving an oadrDistributeEvent from the VTN.

Figure 4 EiEvent PUSH Pattern

For the PULL case the VEN requests events by sending an oadrPoll to the VTN (see section 8.6). The VTN responds with an oadrDistributeEvent. From this point the VEN response is exactly the same as in the PUSH interaction. These sequences are illustrated in Figure 5.

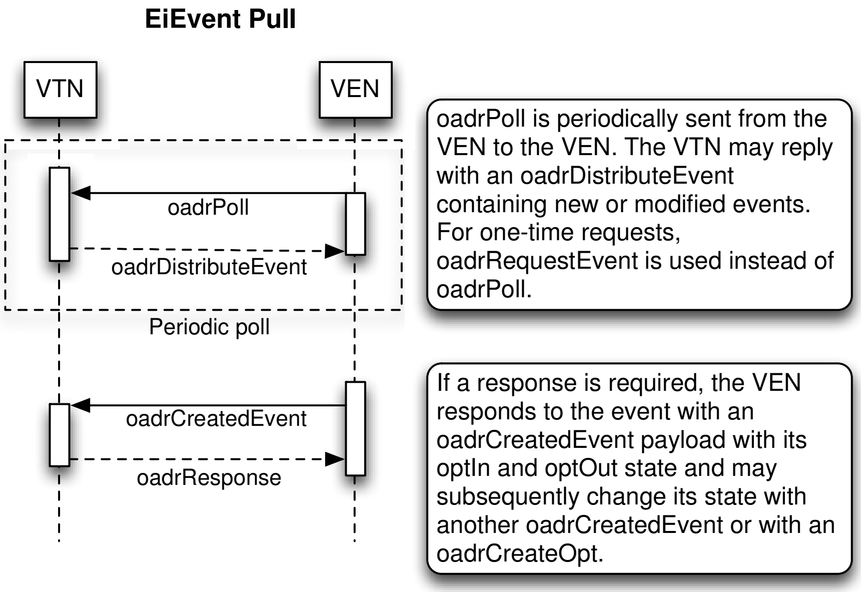

EiEvent Pull
VTN
VEN
oadrPoll
oadrDistributeEvent
Periodic poll
oadrCreatedEvent
oadrResponse
oadrPoll is periodically sent from the VEN to the VEN. The VTN may reply with an oadrDistributeEvent containing new or modified events. For one-time requests, oadrRequestEvent is used instead of oadrPoll.
If a response is required, the VEN responds to the event with an oadrCreatedEvent payload with its optIn and optOut state and may subsequently change its state with another oadrCreatedEvent or with an oadrCreateOpt.

Figure 5 EiEvent PULL Pattern

The details of how these interactions are performed within the context of a specific transport mechanism are covered in Section 9.

When an event requires a response, an initial oadrCreatedEvent is always sent from the VEN to the VTN. If a given program allows a VEN to later change its opt-state during an event it may do so by issuing subsequent oadrCreatedEvent (or oadrCreateOpt) message containing the new state for a given event. Detailed descriptions of VTN and VEN event processing are given in the following paragraphs.

Note that for both PUSH and PULL operations, an oadrDistributeEvent payload will always contain all events applicable to the VEN it is communicating with.

Events are conveyed in the oadrDistributeEvent payload using one or more oadrEvent elements. The oadrDistributeEvent payload has the following components:

- A requestID to uniquely identify this request and any contained events.
- A vtnID identifying the VTN sending the event.
- Zero or more oadrEvent elements.

The requestID uniquely identifies the request and any contained events. Its value is set by the VTN using whatever scheme they desire, including using the same value in every request if its use is not needed by the VTN. The receiving VEN must use this requestID in the oadrCreatedEvent event responses.

oadrEvent Description

oadrEvent elements describe individual events, signal values, and time periods that apply to signals. Each oadrEvent has an eiEvent element containing detailed event information and an oadrResponseRequired element that controls whether a VEN must respond with an oadrCreatedEvent.

The responseRequired field indicates how the VEN must respond to contained events. A value of “always” indicates that the VEN must reply to each event whether or not the event state (eventID, modificationNumber) has changed. A value of “never” implies a “broadcast” event and the VEN must not send any responses in this case.

The eiEvent eventDescriptor has the following fields:

- eventID – a unique ID for this event. The ID is unique within the context of a VEN.
- modificationNumber – A sequence that starts at zero and is incremented by 1 each time the VTN modifies the event.
- modificationDateTime – The time the event has been modified (only in 2.0b payloads)
- modificationReason – The reason why the event has been modified (only in 2.0b payloads)
- priority – An indication of the event priority with 0 being no priority
- marketContext – Identifies a particular program or application defined grouping that pertains to an event.
- createdDateTime – The time the payload containing the event was created. Note that payloads are not necessarily created each time they are transmitted as they may be buffered and only recreated if the event is modified. Any change of the payload of the event requires an update of createdDateTime.
- eventStatus – The status of the event, indicating if the event is “near”, “far”, “active” or “cancelled”.
- testEvent – If not false, indicates this is a test event.
- vtnComment – Arbitrary comment provided by the VTN.

A single eiActivePeriod defines the start time and duration of the event. The start time is defined by an ISO 8601 time descriptor and an ISO 8601 duration string specifies the duration. Note that RFC 5545 (iCalendar) does not support the representation of years and months in the duration string that is otherwise similar to the one specified in ISO 8601. OpenADR 2.0 implementations are required to support the ISO 8601 representation (including support for fractional seconds).

Event start times can be randomized using the tolerance object in the eiActivePeriod. The subelement startafter defines a randomization time window used by the VEN to select a random value that is added to the start time of the event. If the start time of a one hour event is 3:00pm and the randomization window is 5 minutes, if the VEN selected 3 minutes as the random value then the event would start at 3:03pm and would end at 4:03pm

The event signals that get applied over the entire active period are defined in an eiEventSignals element. This element contains one or more eiEventSignal elements (for OpenADR 2.0a at most one), each with a sequence of durations, the sum of which must equal the full duration of the active period. Each signal element contains a signalType such as level or price. The signalPayload contains the value of the signal according to table Table 1 (for the B profile), and relative values of “normal”, “moderate”, “high”, or “special” (for the A profile) for each duration.

Figure 6 depicts the different time periods of an event. EIv1.0 specifies the following elements as:

Ramp Up Period:

> Period at the beginning of the Active Interval expressed as a Duration, during which a VEN moves from its former state to its requested state. If negative, then the Ramp Up occurs within the bounds of the Active Interval, i.e., it starts at the same moment as the Active Interval. If there is no Ramp Up Period, then all other rules are processed as if there were a Ramp Up Period of zero length.

Recovery Period:

Period at the end of the Active Interval expressed as a Duration during which the effect of the response may be reversed while the system returns to its base state. For example, a system that reduces energy use during an Event by raising the air temperature may use additional energy during the recovery period while cooling the air to the normal setting. If negative, then the Recovery Period occurs within the bounds of the Active Interval, i.e., it ends at the same moment as does the Active Interval.

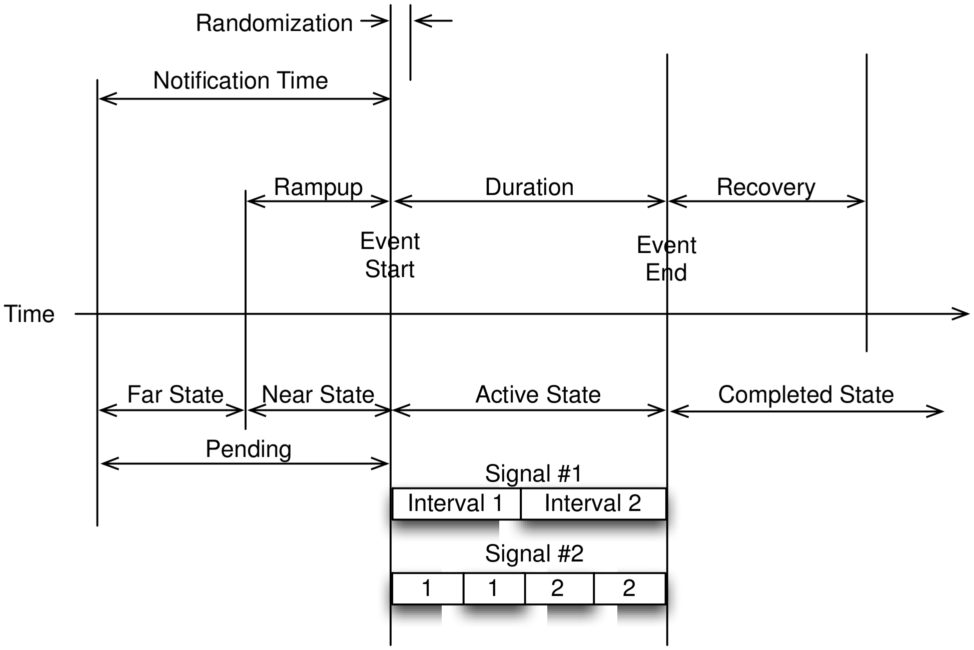


Figure 6 Time intervals of an event

The EiTarget may be used to provide information to explicitly specify the entities that apply to events. The B profile also supports signal specific EiTargets. An EiTarget may contain one venID, and one or more targets such as groupIDs, resourceIDs, or partyIDs (many others available in the B profile). These may be used, for example, when a VEN is acting as an aggregator and has multiple resources behind it. Exactly how these IDs are handled is beyond the scope of this specification and must be dealt with by provisioning at the VEN and VTN. If the EiTarget element is empty, the VEN must assume that it is the primary and only resource targeted by the events.

oadrCreatedEvent Description

When one or more received events require a response, the VEN creates and populates an oadrCreatedEvent element and posts it to the VTN. The eiResponse element contains an application level responseCode and responseDescription and a requestID. An eiResponses element contains one or more eventResponse elements corresponding to each event. These are matched to specific events using the qualifiedEventID, which contains an eventID and modificationNumber. The optType may have a value of “optIn” or “optOut” to indicate the VENs disposition for a given event.

An initial oadrCreatedEvent response must be sent for each event requiring a response. Subsequent oadrCreatedEvent messages may also be sent to change the opt-state of a VEN when this is allowed for a given marketContext.

The grouping of events in an oadrCreatedEvent is completely up to the VEN and does not necessarily correspond to the grouping of events in an oadrDistributeEvent. The VEN is free to send its opt-statuses by transmitting one oadrCreatedEvent payload per event or group multiple event responses into a single oadrCreatedEvent payload.

8.1.1 Data Model

The OpenADR Alliance has its own schema that defines Alliance specific extensions and references elements from Energy Interoperation and related schemas.

For each schema referenced by Energy Interoperation and used by OpenADR profiles, the OpenADR Alliance created a separate subset schema XSD file. These schema subsets will maintain the same schema hierarchical element relationships, ordering of elements, mandatory cardinality, type definitions, and restrictions. However, the subset schemas may not use the same structural elements, such as abstract types and substitution groups, as the Energy Interoperation and related schemas. The Alliance will associate namespace prefixes with the same URIs used by the Energy Interoperation and related schemas.

URIs referenced in the OpenADR schema can be mapped to either the Alliance subset schema files or the Energy Interoperation 1.0 standard and related schema files. Individual components of the OpenADR payloads will successfully validate against the OpenADR schema and the OASIS Energy Interoperation 1.0

8.1.2 UML Models

Figure 7 shows the OpenADR 2.0 EiEvent Service using the OASIS EI 1.0 standard. The green highlights are the elements that apply to OpenADR 2.0a with an indication if these elements are mandatory (M) or optional (O), and the additional element that are needed for OpenADR 2.0b in red.

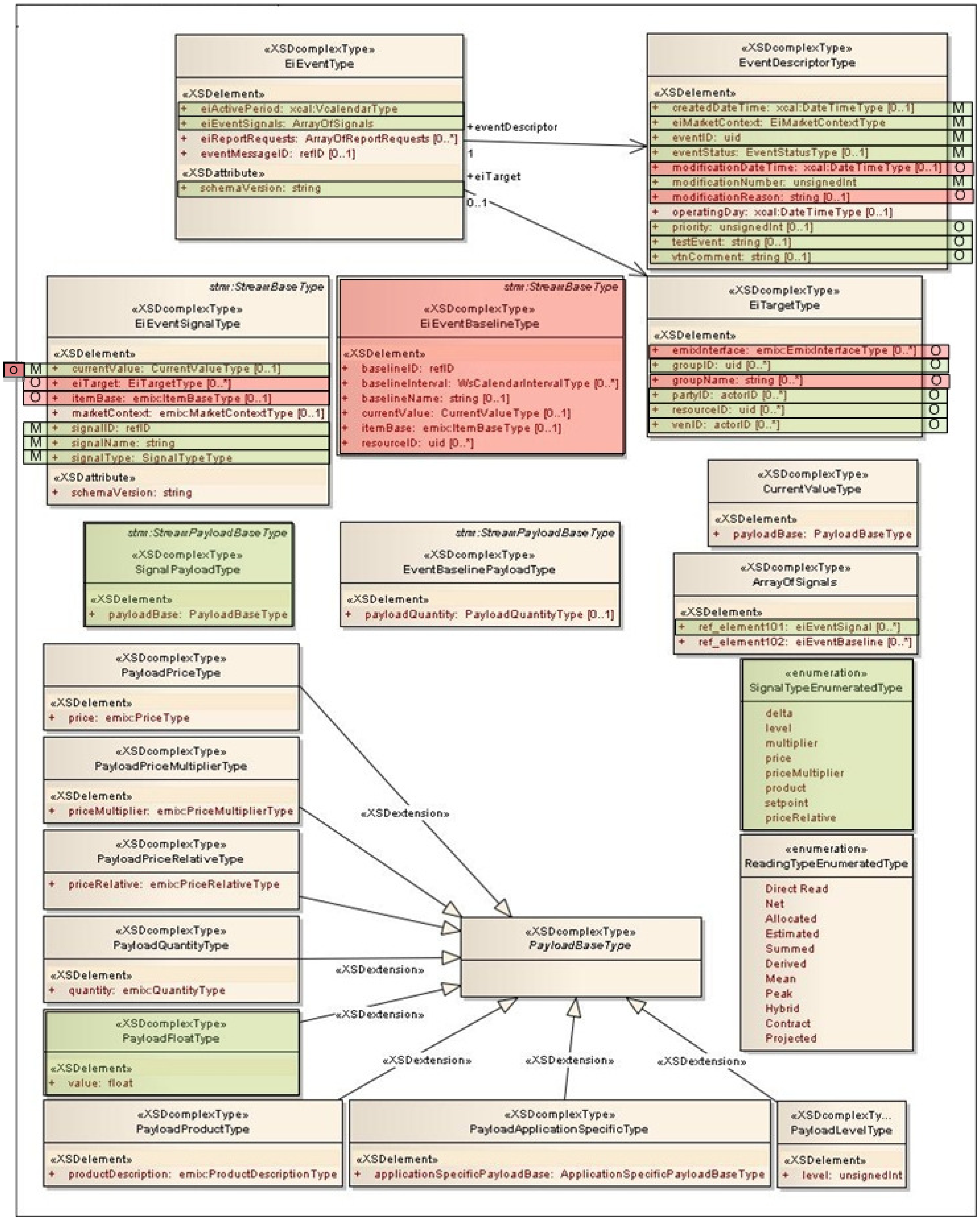

«XSDcomplexType»
EiEventType
«XSDelement»
+ eiActivePeriod: xcal:VcalendarType
+ eiEventSignals: ArrayOfSignals
+ eiReportRequests: ArrayOfReportRequests [0..*]
+ eventMessageID: refID [0..1]
«XSDattribute»
+ schemaVersion: string
+eventDescriptor
1
+eiTarget
0..1
«XSDcomplexType»
EventDescriptorType
«XSDelement»
+ createdDateTime: xcal:DateTimeType [0..1]
M
+ eiMarketContext: EiMarketContextType
M
+ eventID: uid
M
+ eventStatus: EventStatusType [0..1]
M
+ modificationDateTime: xcal:DateTimeType [0..1]
O
+ modificationNumber: unsignedInt
M
+ modificationReason: string [0..1]
O
+ operatingDay: xcal:DateTimeType [0..1]
+ priority: unsignedInt [0..1]
O
+ testEvent: string [0..1]
O
+ vtnComment: string [0..1]
O
strm:StreamBaseType
«XSDcomplexType»
EiEventSignalType
«XSDelement»
O
M
+ currentValue: CurrentValueType [0..1]
O
+ eiTarget: EiTargetType [0..*]
O
+ itemBase: emix:ItemBaseType [0..1]
+ marketContext: emix:MarketContextType [0..1]
M
+ signalID: refID
M
+ signalName: string
M
+ signalType: SignalTypeType
«XSDattribute»
+ schemaVersion: string
strm:StreamBaseType
«XSDcomplexType»
EiEventBaselineType
«XSDelement»
+ baselineID: refID
+ baselineInterval: WsCalendarIntervalType [0..*]
+ baselineName: string [0..1]
+ currentValue: CurrentValueType [0..1]
+ itemBase: emix:ItemBaseType [0..1]
+ resourceID: uid [0..*]
«XSDcomplexType»
EiTargetType
«XSDelement»
+ emixInterface: emix:EmixInterfaceType [0..*]
O
+ groupID: uid [0..*]
O
+ groupName: string [0..*]
O
+ partyID: actorID [0..*]
O
+ resourceID: uid [0..*]
O
+ venID: actorID [0..*]
O
«XSDcomplexType»
CurrentValueType
«XSDelement»
+ payloadBase: PayloadBaseType
strm:StreamPayloadBaseType
«XSDcomplexType»
SignalPayloadType
«XSDelement»
+ payloadBase: PayloadBaseType
strm:StreamPayloadBaseType
«XSDcomplexType»
EventBaselinePayloadType
«XSDelement»
+ payloadQuantity: PayloadQuantityType [0..1]
«XSDcomplexType»
ArrayOfSignals
«XSDelement»
+ ref_element101: eiEventSignal [0..*]
+ ref_element102: eiEventBaseline [0..*]
«enumeration»
SignalTypeEnumeratedType
delta
level
multiplier
price
priceMultiplier
product
setpoint
priceRelative
«enumeration»
ReadingTypeEnumeratedType
Direct Read
Net
Allocated
Estimated
Summed
Derived
Mean
Peak
Hybrid
Contract
Projected
«XSDcomplexType»
PayloadPriceType
«XSDelement»
+ price: emix:PriceType
«XSDcomplexType»
PayloadPriceMultiplierType
«XSDelement»
+ priceMultiplier: emix:PriceMultiplierType
«XSDcomplexType»
PayloadPriceRelativeType
+ priceRelative: emix:PriceRelativeType
«XSDcomplexType»
PayloadQuantityType
«XSDelement»
+ quantity: emix:QuantityType
«XSDcomplexType»
PayloadFloatType
«XSDelement»
+ value: float
«XSDextension»
«XSDextension»
«XSDextension»
«XSDcomplexType»
PayloadBaseType
«XSDextension»
«XSDextension»
«XSDextension»
«XSDcomplexType»
PayloadProductType
«XSDelement»
+ productDescription: emix:ProductDescriptionType
«XSDcomplexType»
PayloadApplicationSpecificType
«XSDelement»
+ applicationSpecificPayloadBase: ApplicationSpecificPayloadBaseType
«XSDcomplexTy...
PayloadLevelType
«XSDelement»
+ level: unsignedInt

Figure 7 EiEvent UML (green – A profile; red – B profile additions)

8.2 Differences between OpenADR 2.0a and 2.0b Event Mechanism

The OpenADR 2.0b profile supports additional functionalities not supported by 2.0a (see also section 7.1). The 2.0b profile adds support for the following capabilities, mostly relating to event signals:

- Event signal type is not limited to SIMPLE. The full set of signals described in section 8.2.2 is supported.
- Events may contain multiple event signals.
- Event signals may contain EiTarget elements, distinct from the event-level EiTarget. However, the target types must be constrained to endDeviceAsset with the enumerated values for device type shown in the conformance rules.
- Event signals may contain baseline elements. The rules for baseline overall duration and intervals are the same as those between the event active period and event signal intervals.
- The event description may contain modificationDateTime and modificationReason elements.
- The EiOpt service can be used to send schedules from the VEN to the VTN, in addition to oadrCreatedEvent from 2.0a.
- The oadrPoll mechanism is used in a simple HTTP PULL exchange to model the behavior of a PUSH implementation. For more information refer to section 8.6.

8.2.1 Event Targets and Resources

While the A Profile supported event targets (EiTarget), the B Profile includes several more target types, and also allows device class targeting to be applied at the signal level. A VEN is a communication endpoint that represents one or more logical resources (individual shedable loads, endpoint equipment, meters, etc.). Event targets select which specific VEN resources the event applies to. If an event target is not specified, the VEN should assume that it applies to all of its resources. How resources are assigned properties (location, pnode associations, resourceIDs, groupIDs, etc.) is outside the scope of the specification and is up to deployment configurations in the VTN and VEN. However, if a VEN receives an event target that it is not configured for, it should reject the message with the appropriate error code described in section 8.7.

8.2.2 OpenADR 2.0b Signal Definitions

Profile B allows for a more diverse set of signals in the event messages. The eiEventSignal:signalName, eiEventSignal:signalType, and eiEventSignal:itemBase attributes are used to describe the signal. Table 1 lists standard pre-defined signals that may be used. The purpose of the pre-defined signals is to establish a common set of signals and their attributes for the purposes of interoperability. For compliance purposes it is not a requirement that a VTN or VEN support all the signals listed in the table below.

Furthermore specific deployments are free to define their own custom signals beyond what are defined in the table below, but there are no requirements for compliance purposes that any VTN's or VEN's support such signals.

The value of the eiEventSignal:signalID attribute is deployment specific and primarily used to differentiate signals in cases where there may be multiple signals of the same type.

Table 1 Signals

Signal Category	Name (signalName)	Type (signalType)	units (itemBase)	Allowed Values	Description
Simple levels	SIMPLE	level	None	0,1,2,3	Simple levels
Price of electricity	ELECTRICITY_PRICE	price	currency/kWh	any	This is the cost of electricity expressed in absolute terms
	ELECTRICITY_PRICE	priceRelative	currency/kWh	any	This is a delta change to the existing price of electricity
	ELECTRICITY_PRICE	priceMultiplier	None	any	This is a multiplier to the existing cost of electricity
Price of energy	ENERGY_PRICE	price	currency/kWh	any	This is the cost of energy expressed in absolute terms
	ENERGY_PRICE	priceRelative	currency/kWh	any	This is a delta change to the existing price of energy
	ENERGY_PRICE	priceMultiplier	None	any	This is a multiplier to the existing cost of energy
Demand charge	DEMAND_CHARGE	price	currency/kW	any	This is the demand charge expressed in absolute terms
	DEMAND_CHARGE	priceRelative	currency/kW	any	This is a delta change to the existing demand charge
	DEMAND_CHARGE	priceMultiplier	None	any	This is a multiplier to the existing demand charge
Customer bid levels	BID_PRICE	price	currency/XX (2)	any	This is the price that was bid by the resource
	BID_LOAD	setpoint	powerXXX (1)	any	This is the amount of load that was bid by a resource into a program
	BID_ENERGY	setpoint	energyXXX (1)	any	This is the amount of energy from a resource that was bid into a program
Used to dispatch storage resources	CHARGE_STATE	setpoint	energyXXX (1)	any	This is used to either charge or discharge a certain amount of energy from a storage resource until its charge state reaches a certain level.
	CHARGE_STATE	delta	energyXXX (1)	any	This is the delta amount of energy that should be contained in a storage resource from where it currently is.
	CHARGE_STATE	multiplier	None	0.0 < 1.0	This is the percentage of full charge that the storage resource should be at.

These instructions are used to set the load to values that can be expressed in terms of the desired load	LOAD_DISPATCH	setpoint	powerXXX (1)	any	This is used to dispatch loads to a specific amount
	LOAD_DISPATCH	delta	powerXXX (1)	any	This is used to dispatch loads to some offset from an agreed upon baseline. Note that the baseline may be the current power consumption.
	LOAD_DISPATCH	multiplier	None	any	This is used to dispatch loads as some multiple of power against some agreed upon baseline. Note that the baseline may be the current power consumption.
	LOAD_DISPATCH	level	powerXXX	integer value from -10 to +10	This is used to specify the load in terms of discrete levels.
These instructions are used to set the load control to values that are relative to the load controller and its output capacity. This does not require the VTN or the VEN knowing precisely what the load consumption level is, but are expressed in ways that the VTN can know that the signal values will either increase or decrease the load consumption regardless of the specific type of device that is performing the load control. These can be used for some aspects of direct load control by mapping these general instructions to specific load control commands in the VEN without the VTN needing to know precisely what device may be con-	LOAD_CONTROL	x-loadControlCapacity	None	0 - 100% (0.0 - 1.0)	This is an instruction for the load controller to operate at a level that is some percentage of its maximum load consumption capacity. This can be mapped to specific load controllers to do things like duty cycling. Note that 1.0 refers to 100% consumption. In the case of simple ON/OFF type devices then 0 = OFF and 1 = ON.
	LOAD_CONTROL	x-loadControlLevelOffset	None	integer value, Positive or negative	Discrete integer levels that are relative to normal operations where 0 is normal operations. There is no requirement to link the setpoints to specific load consumption values, but the intention is that the higher the setpoint the less load is consumed. Note that these are controller set points that can be mapped at the VEN side to something as simple as thermostat temperature set point offsets.

suming the signal.	LOAD_CONTROL	x-loadControlSetpoint	None	any value	Load controller set points. There is no requirement to link the setpoints to specific load consumption values. Note that these are generic controller set points and can be mapped at the VEN side to something as simple as specific thermostat temperature set points.
	LOAD_CONTROL	x-loadControlPercentOffset	None	any percentage, both positive and negative	Percentage change from normal operations. The percentage does not refer to specific load consumptions values, but to load control operation levels. The lower the percentage the less load is consumed.

(1) The XXX in the table represents Real, Apparent, and Reactive versions of power or energy
(2) The XX in the table represents the currency unit, such as USD

8.3 OpenADR 2.0b Report Service

8.3.1 Introduction

This section describes the reporting service for the OpenADR 2.0b profile. For detailed report content please seeAnnex A.

8.3.1.1 Report Types

The OpenADR Alliance supports several well-known report types. Each report type is intended to represent a certain set of reporting functionality that is supported by either a VEN or a VTN that claims to support that type. Figure 8 gives the taxonomy of the OpenADR report types.

For compliance purposes VTNs and VENs are not required to support all the different types of reports (see conformance rule 510). Furthermore for specific deployments there may be reports used that do not conform to any of the standard Alliance report profiles, but these are not relevant for compliance purposes.

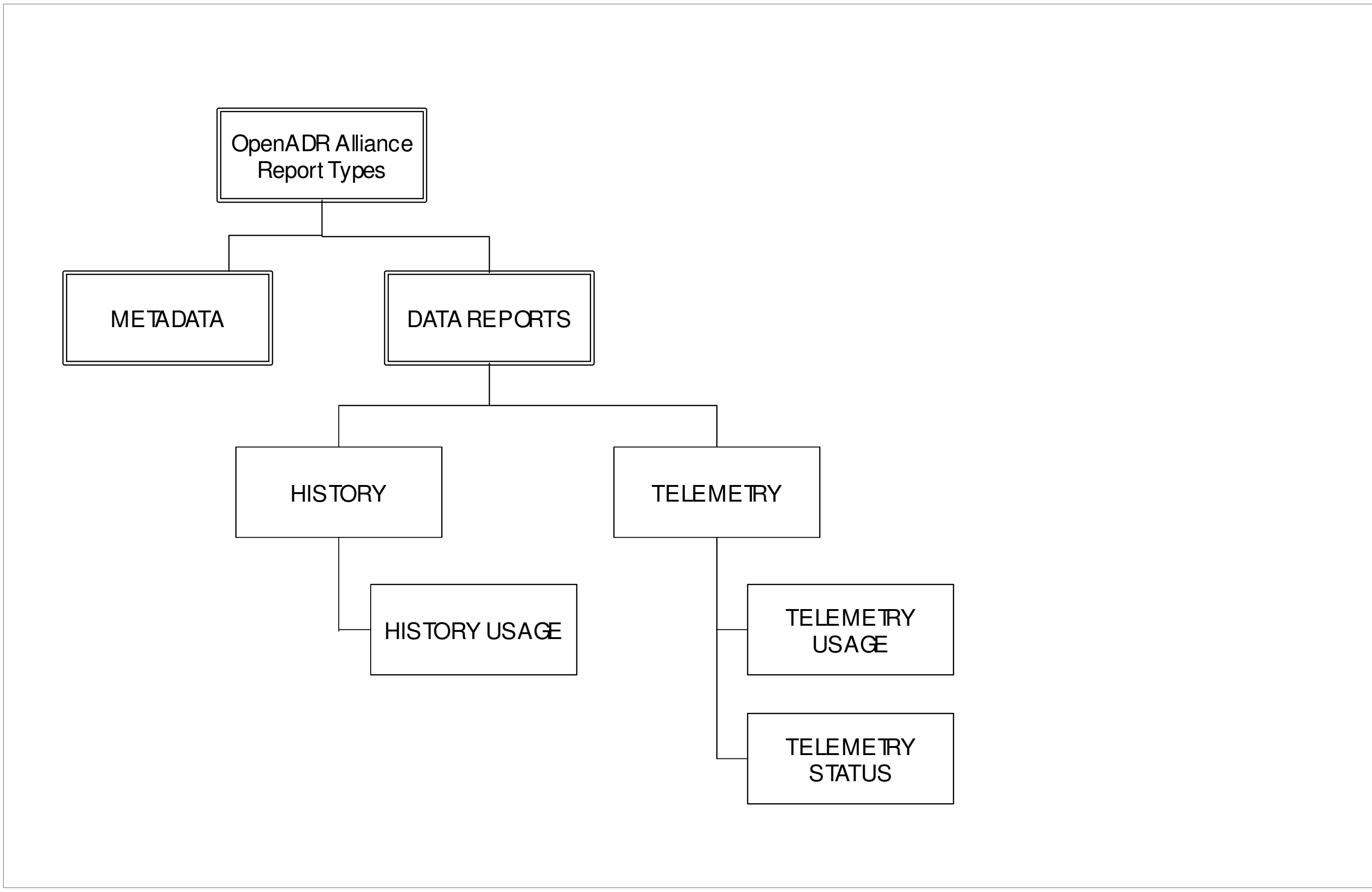


Figure 8 Report types

Each of the report types is described in more detail below.

METADATA – This report type is used to specify reporting capabilities. It is exchanged as part of the report registration process that is described in section 8.3.2.1. The METADATA report can contain a specification for one or more type of reports, each report having its own set of report descriptors and specifications. Each report in the METADATA report has a reportSpecifierID that is used in subsequent interactions to refer to that report specification. Each report specification in the METADATA report will use the reportName attribute to indicate which of the well-known report profiles it is referring to. All VTN and VEN implementations that conform to profile B must support the METADATA report and in the case where a VTN or VEN does not support any reporting capabilities, for compliance purposes it must still support sending the METADATA report message in order to portray the fact that it does not support any reporting capabilities.

DATA REPORTS (non-metadata reports) – These reports are used to report actual data that may be measured or calculated. The core element of a Data Report is the so called "data point". A data point represents a certain quantity that may be measured or calculated as part of a report. Each data point has attributes such as units, etc. A Data Report may contain one or more data points. The METADATA report will contain a report with a set of data points that can appear in that report and when an actual report is requested the set of data points that should appear in the report are specified by the requesting party. Each data point is represented in the schema with the rID attribute. For example, assume a VEN can measure both energy and power as two separate data points. The VEN has the choice of either specifying that it can generate two separate reports, each with a single data point or it may specify that it can generate a single data report with two data points. The VTN would then make the appropriate report request to get the data.

There are several sub-categories of Data Reports as described below.

- HISTORY DATA REPORTS – This is a type of data report in which the history of the data point values is logged and can be subsequently requested. These include the following specific types:
  - HISTORY_USAGE – these are logs of usage data that are typically logged by VEN's and can be queried by the VTN

- TELEMETRY DATA REPORTS – The term telemetry in the context of OpenADR refers to data that is reported periodically in real time and includes the following specific report types:
  - TELEMETRY_USAGE – this is usage data that is periodically reported from the VEN to the VTN in real time.
  - TELEMETRY_STATUS – This is the status of a resource, which may be periodically reported from the VEN to the VTN.

8.3.2 Core Reporting Operations

The reporting service supports the exchange of reports between the VEN and VTN and vice versa. All report interactions between the VEN and the VTN are built upon the following core operations.

- Registering Reporting Capabilities
- Requesting Reports
- Sending Reports
- Canceling Reports

This section describes the logical payload exchanges that support each of these operations.

8.3.2.1 Register Reporting Capabilities

This general use case describes how one party's reporting capabilities are registered with the other party. Report registration is performed after completion of party registration, both from the VTN to the VEN and from the VEN to the VTN. In addition, any party may send report registrations at any time after the initial registration.

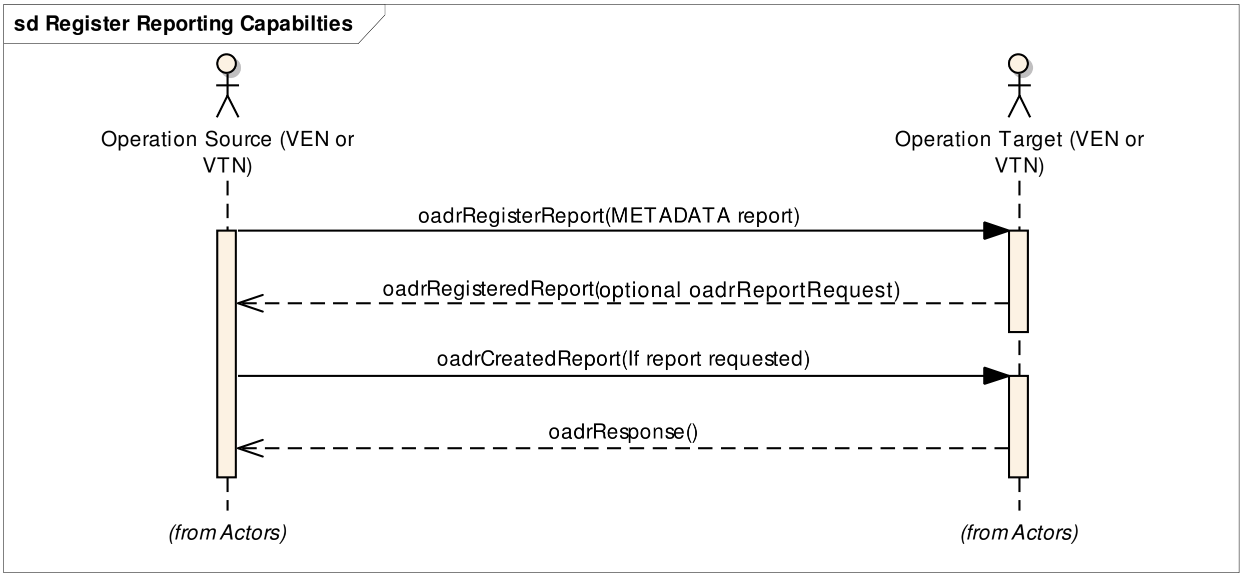

sd Register Reporting Capabilties
Operation Source (VEN or VTN)
Operation Target (VEN or VTN)
oadrRegisterReport(METADATA report)
oadrRegisteredReport(optional oadrReportRequest)
oadrCreatedReport(If report requested)
oadrResponse()
(from Actors)
(from Actors)

Figure 9 Interaction diagram: Register Reporting Capabilities

In this use case, depicted in Figure 9, the source party sends its reporting capabilities to the target party. The source party's reporting capabilities are specified using a special well-known report profile called the METADATA report, which is exchanged using the same schema as any other report.

The METADATA report contains a collection of all the different types of reports that can be sent by the source party (VEN or VTN) that is generating the report. Each reporting capability specified in the METADATA report is uniquely identified by using the reportSpecifierID attribute, which is generated by the source party when registering the report. The reportSpecifierID will allow the target party to make subsequent requests for that specific report.
Each report specified in the METADATA report is a specification for the elements and attributes that may appear in that report and is based upon one the well-known report profiles. By specifying the meta data for the report as part of this registration process, the actual reports that may be exchanged in the future need only refer to the reportSpeciferID along with the actual data without the need to send all the other report description attributes such as units, etc.

The interaction proceeds using the following steps:

(1) The source party first sends its reporting capabilities to the target party by using the oadrRegisterReport payload. In general the oadrRegisterReport payload is the same as the oadrUpdateReport payload except that it contains a METADATA report. The source party sends the special well-known report of type METADATA using the oadrReport schema as described above.
(2) The target party responds with the oadrRegisteredReport payload. The target party's response may contain an oadrReportRequest object requesting which reports the source party should generate. If there are reports that the target party knows that it wants to receive from the source party then it should make those requests as part of this step. This is similar to requesting a report using oadrCreateReport as specified in section 8.3.2.2.
(3) If the target party requests that the source party create any reports as part of step (2) then the source party responds with the oadrCreatedReport payload.

In essence step (2) and (3) are equivalent to and use the same data structures as the interaction in which the target party requests reports from the source party using oadrCreateReport, as specified in section 8.3.2.2.

Every exchange of the METADATA report must supply all the reporting capabilities of the source party (VTN or VEN) and will therefore supplant any previously exchanged METADATA report. Furthermore the sending of the METADATA report implicitly cancels the sending of all reports that the source party may have previously requested from the target party.

Key elements included in the metadata report payloads include:

- reportSpecifierID: A unique identifier for the reporting capability, used in subsequent requests
- rid: A unique identifier for each data point offered by the report
- Duration: the amount of time that data can be collected (or has been collected thus far for history)
- SamplingRate.oadrMinPeriod = maximum sampling frequency
- SamplingRate.oadrMaxPeriod = minimum sampling frequency
- SamplingRate.onChange = whether or not data is sampled as it changes

If the target party does not request any reports as part of step (2) then the source party should assume that there are no reports to be generated and sent to the target party. Therefore if there are any reports that the target party has previously requested from the source party then it must make those requests again as part of step (2).

The source party can therefore use this interaction at any time to request that the target party notify the source party of any reports that it should create and send. This provides a mechanism for the source and target party to synchronize both the reporting capabilities of the source party and the report requests of the target party.

8.3.2.2 Request Reports

This use case, depicted in Figure 10, shows how one party may request reports from the other party. Note that any reports that are being requested must correspond to one of the reports that were previously specified in the METADATA report that was previously sent by the other party.

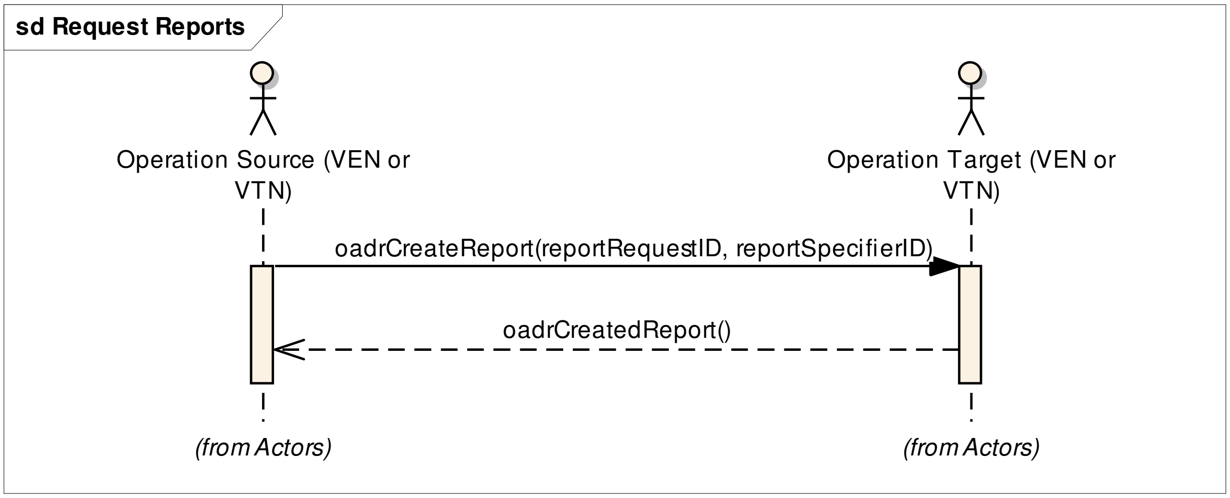

sd Request Reports
Operation Source (VEN or VTN)
Operation Target (VEN or VTN)
oadrCreateReport(reportRequestID, reportSpecifierID)
oadrCreatedReport()
(from Actors)
(from Actors)

Figure 10 Interaction diagram: Request Reports

The source party requests a report from the target party by using the oadrCreateReport payload. That payload contains a set of reportSpecifierID's that correspond to report capabilities in the METADATA report that was previously sent by the target party as part of the previously described oadrRegisterReport interaction (refer to section 8.3.2.1). As part of the report request the source party specifies a set of reportRequestID's that are used in subsequent operations on this report request.

In short the reportSpecifierID is generated by the target during report registration (as specified in section 8.3.2.1) to uniquely identify the report contents of a METADATA report, while the reportRequestID is generated by the source party of an oadrCreateReport (or oadrReportRequest object in an oadrRegisteredReport during registration) and associates the reportSpecifierID with a specific report request. For example, the source requesting a report may select only a certain subset of all data points (i.e., rIDs) that have been registered in the METADATA report identified by reportSpecifierID.

All subsequent reports sent from the target party that are in response to this request use the reportRequestID to identify the report.
The response to the oadrCreateReport payload is the oadrCreatedReport payload.

Note that the source party can only send the oadrCreateReport after the target party has sent its METADATA report as part of the reporting registration process. The exception to this is the METADATA report, which may be requested at any time by using the well-known string “METADATA” as the reportSpecifierID.

While the reportSpecifierID dictates the type of data that may appear in a report there are some reporting parameters that are specified by the source party when making this request including the following:

- The specific data items to appear in the report as specified by the rIDs from the corresponding METADATA report. The rID attribute is an identifier in the METADATA report that is associated with a specific type of value that may appear in the report. This is sometimes referred to as a data point and allows multiple data points to exist in a single report. The rID's allow the source party to request a subset of possible data points that should be reported by the target party.
- Whether this is a one-time report or a recurring data stream.
- If it is a recurring data stream the following temporal parameters are also specified: The time frame over which the report should be generated (i.e., next month, indefinitely, etc.), the time intervals at which the data points will be reported, and the frequency at which the report should be generated.
- If it is a onetime history or forecast report then the request specifies the span of time that the report should cover.
  The meaning of dtstart in the report request defines the start time of the reporting period while the dtstart in the returned report indicates the start time of data collection period.

For periodic telemetry reports containing point data, if granularity and duration are the same the report is sent immediately after the request. If granularity is smaller than duration, the delivery of report will be delayed until enough data is collected.

Key elements included in the report request payloads include:

- Start time/Duration: when reporting should commence using granularity as sampling frequency. If Duration == 0, forever

- Granularity: Frequency of reported data. Value must be between SamplingRate.oadrMinPeriod and SamplingRate.oadrMaxPeriod. If 0, the requester is requesting the report to use onChange (if available)
- reportBackDuration: the interval for sending reports (oadrUpdateReport or orderRegisterReport) back to the requester. If reportBackDuration == 0, then it's only a one shot report

See the example report request payloads for more detailed information on the various attributes.

8.3.2.3 Send Reports

This use case defines how the actual reports may be exchanged.

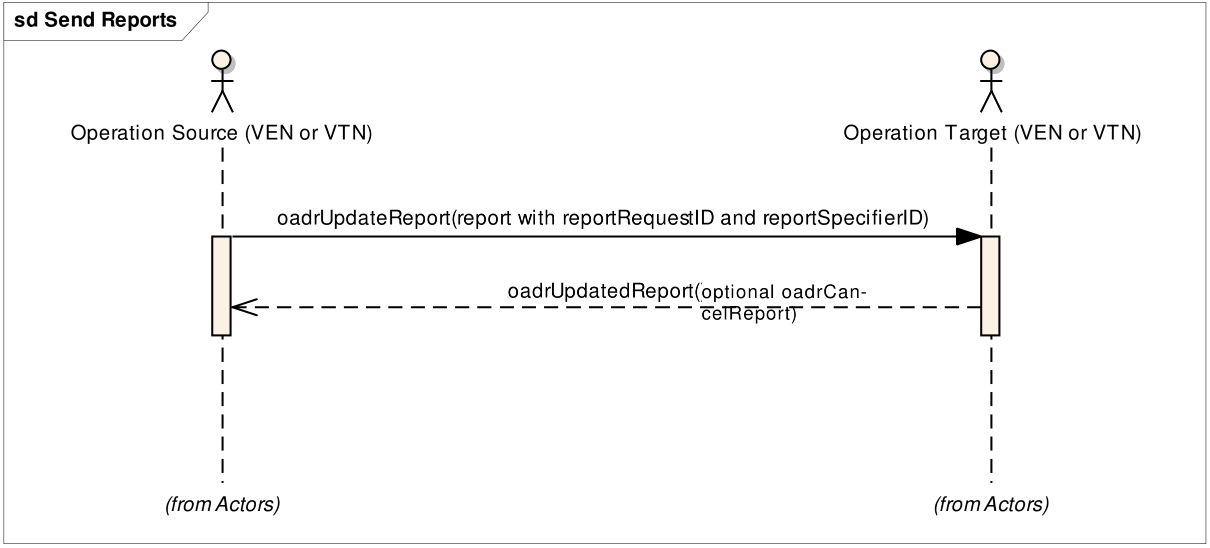

sd Send Reports
Operation Source (VEN or VTN)
Operation Target (VEN or VTN)
oadrUpdateReport(report with reportRequestID and reportSpecifierID)
oadrUpdatedReport(optional oadrCancelReport)
(from Actors)
(from Actors)

Figure 11 Interaction diagram: Send Reports

Figure 11 depicts the source party sending a report to the target party.

This operation can be performed by the source party only after a previous report request interaction is performed by the target party.

This operation uses the same oadrReport object as the report registration operation did, but is used to exchange a report with actual data elements as opposed to the METADATA report used in the registration process.

The reports sent in the oadrUpdateReport payload use the EiReport schema with the reportRequestID as defined by the target party in the previous report request interaction that prompted the sending of this report.

The response sent by the target party uses the oadrUpdatedReport payload to acknowledge receipt of the report. As part of the oadrUpdatedReport response the target party may cancel the sending of any future reports using the optional oadrCancelReport object, which contains a list of reportRequestIDs. There is no confirmation of the cancellation if included in the oadrUpdatedReport payload, but if the report continues to be sent the receiving party should use oadrCancelReport to cancel the report. In the case of periodic report, if reportToFollow is set to true in the oadrCancelReport object by the target party, the source party is expected to send one final additional report.

See the various sample report payloads for more details on the report attributes.

8.3.2.4 Cancel Reports

This interaction, depicted in Figure 12 is used by the source party to cancel ongoing (i.e., periodic) reports that are being generated by the target party.

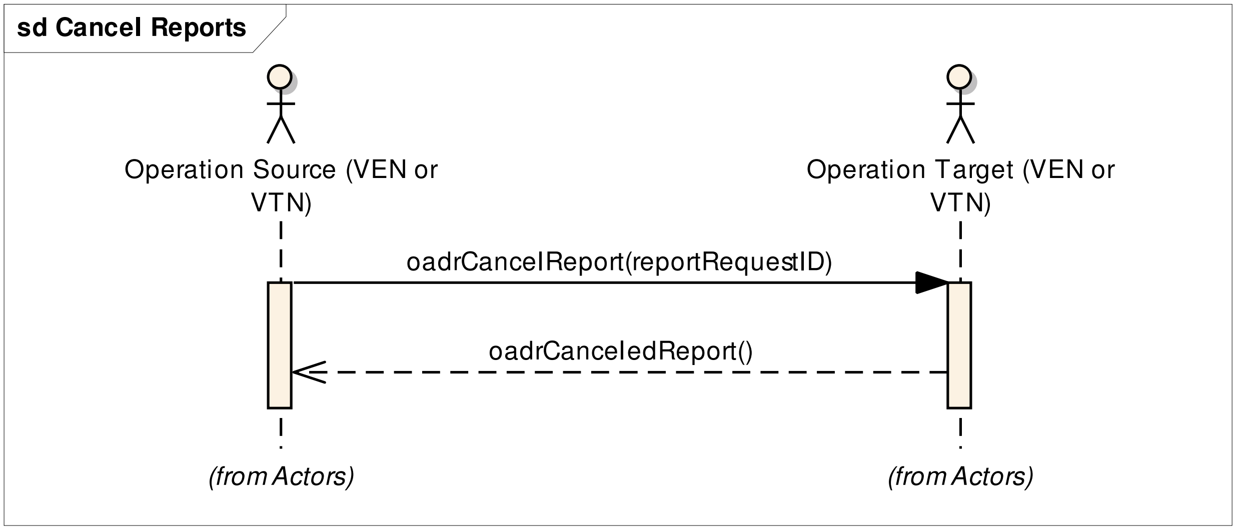

sd Cancel Reports
Operation Source (VEN or VTN)
Operation Target (VEN or VTN)
oadrCancelReport(reportRequestID)
oadrCanceledReport()
(from Actors)
(from Actors)

Figure 12 Interaction diagram: Cancel Reports

The source party uses the oadrCancelReport payload with the appropriate reportRequestIDs that were specified by the source party in a previous request report interaction (section 8.3.2.2). Upon receiving the oadrCancelReport payload the target party stops generating and sending the reports corresponding to the reportRequestIDs.

The response to the oadrCancelReport payload is the oadrCanceledReport payload that is sent by the target party to acknowledge the canceling of the report.

Note that both oadrCanceledReport and oadrCreatedReport return a list of pending reports in the oadrPendingReports object, which includes all reports that are scheduled for future delivery.

When cancellation of periodic report is requested by the target party, if reportToFollow is set to true in oadrCancelReport, the source party is expected to send one final additional report.

8.4 OpenADR 2.0b Registration Service

8.4.1 Service Operations

The OpenADR 2.0 B profile uses the EiRegisterParty service to support in-band registration of VENs with VTNs. The following service operations (listed in Table 2) are supported:

Table 2 EiRegisterParty payloads

Request Payload	Response Payload	Re-questor	Responder
oadrQueryRegistration	oadrCreatedPartyRegistration	VEN	VTN
oadrCreatePartyRegistration	oadrCreatedPartyRegistration	VEN	VTN
oadrCancelPartyRegistration	oadrCanceledPartyRegistration	VEN VTN	VTN VEN
oadrRequestReregestration	oadrResponse	VTN	VEN

Registration may optionally begin with the VEN querying the VTN to determine what profiles, transports, and extensions it may support using the oadrQueryRegistration payload (see Figure 13). This query operation can be initiated using any of the Alliance support transports, however the VEN will need to be configured out of band with the address of the VTN in order to initiate the query. The response to the query is the oadrCreatedPartyRegistration payload and contains information on all the profiles and transports supported by the VTN in addition to any supported extensions to the profile. The information received by the VEN may be used to determine the best configuration to use when formally registering as described in balance of this section.

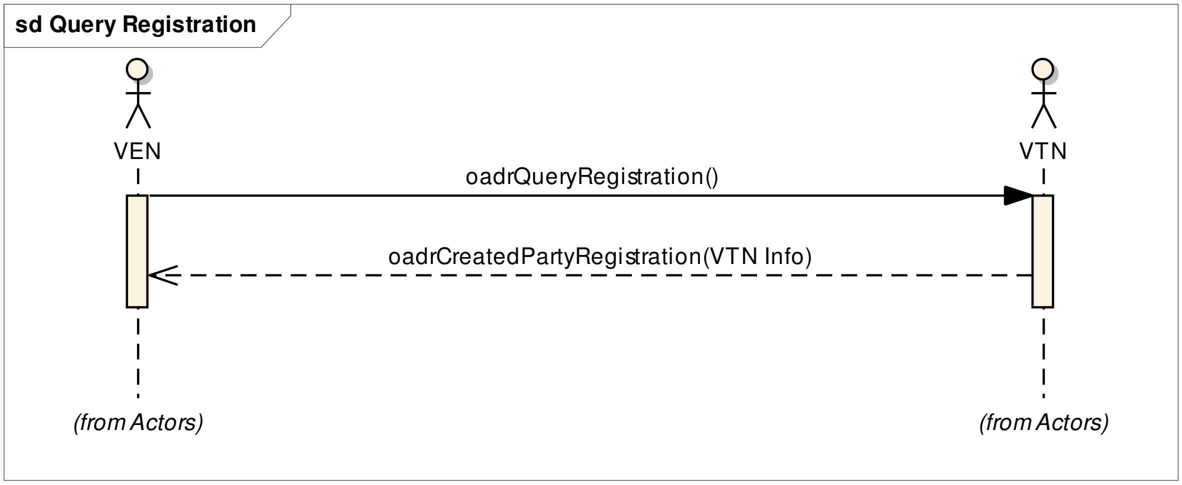


Figure 13 Interaction diagram: Query Registration

Registration is always initiated by the VEN with the oadrCreatePartyRegistration payload (see Figure 14). This payload provides the information on the profile and transport the VEN has decided to use for communication with the VTN, in addition to other registration related information (see Table 3). The VTN responds with an oadrCreatedPartyRegistration containing all the profiles and transports supported by the VTN, IDs, and other registration related information (see Table 4). The VTN returns a registrationID in its response payload, which is used for subsequent operations pertaining to this registration instance.

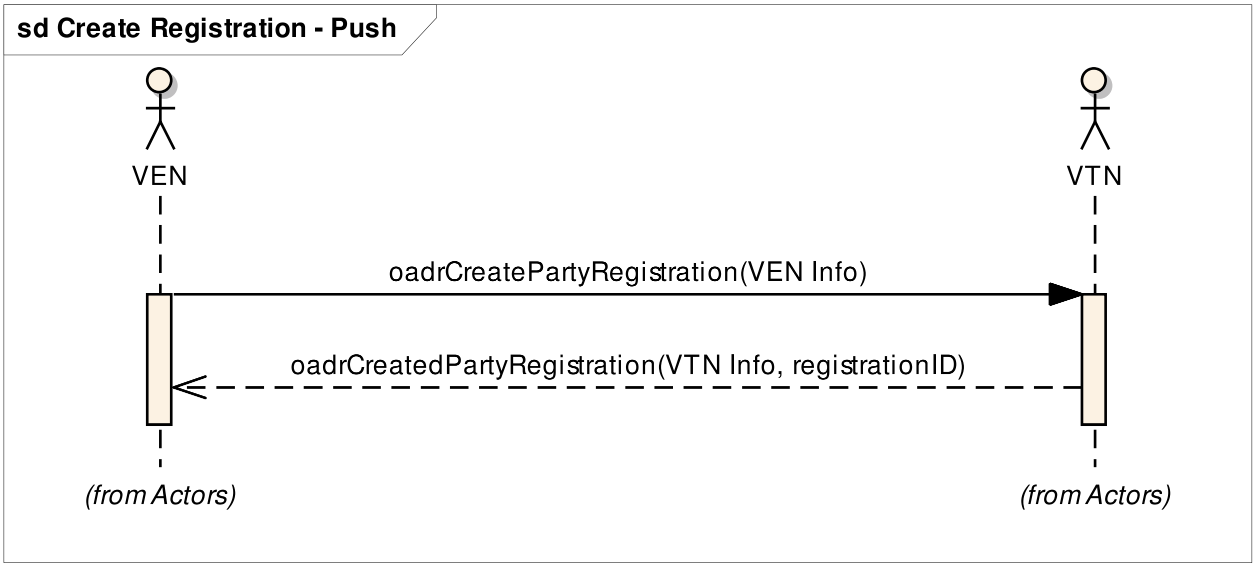

sd Create Registration - Push
VEN
VTN
oadrCreatePartyRegistration(VEN Info)
oadrCreatedPartyRegistration(VTN Info, registrationID)
(from Actors)
(from Actors)

Figure 14 Interaction diagram: Create Registration

If the VEN's registration information changes, the VEN can reregister at any time using the oadrCreatePartyRegistration payload referencing the current registrationID (see Figure 15). The same registrationID will be maintained across reregistrations until one of the parties cancel the registration.

If the VTN's registration information changes, the VTN can request the VEN to reregister using the oadrRequestReregistration payload. The response to this request is an oadrResponse acknowledgement followed by an asynchronous oadrCreatePartyRegistration request from the VEN.

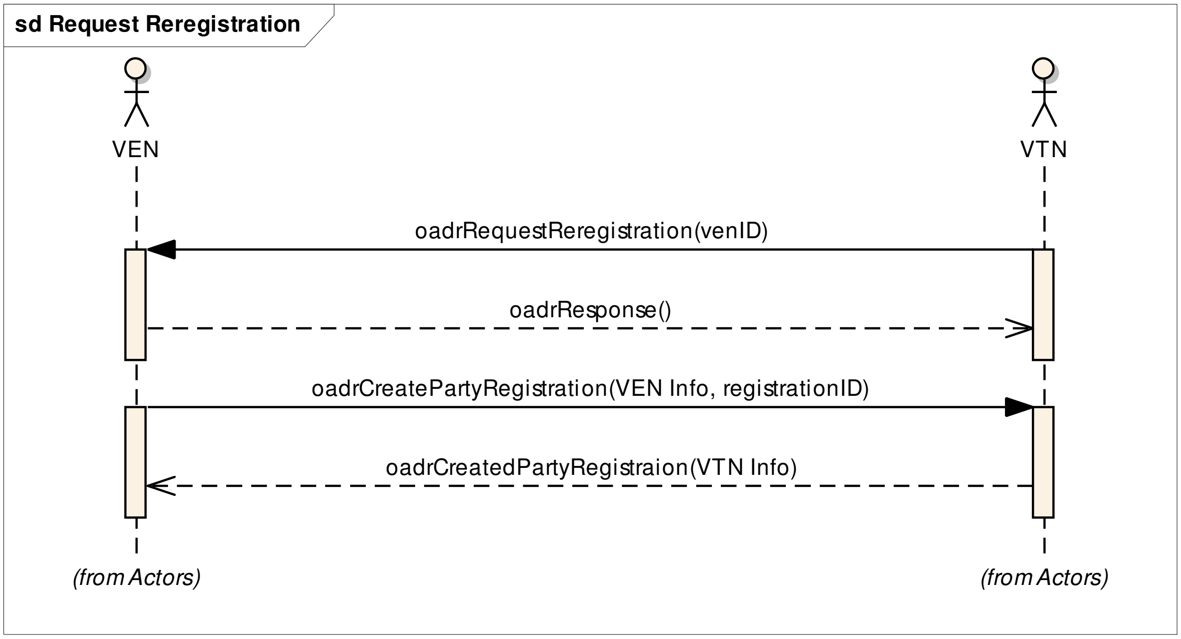

sd Request Reregistration
VEN
VTN
oadrRequestReregistration(venID)
oadrResponse()
oadrCreatePartyRegistration(VEN Info, registrationID)
oadrCreatedPartyRegistraion(VTN Info)
(from Actors)
(from Actors)

Figure 15 Interaction diagram: Request Reregistration

The VTN or VEN may cancel an active registration using the oadrCancelPartyRegistration payload, referencing the registrationID (see Figure 16). The other party responds with an oadrCanceledRegistration payload.

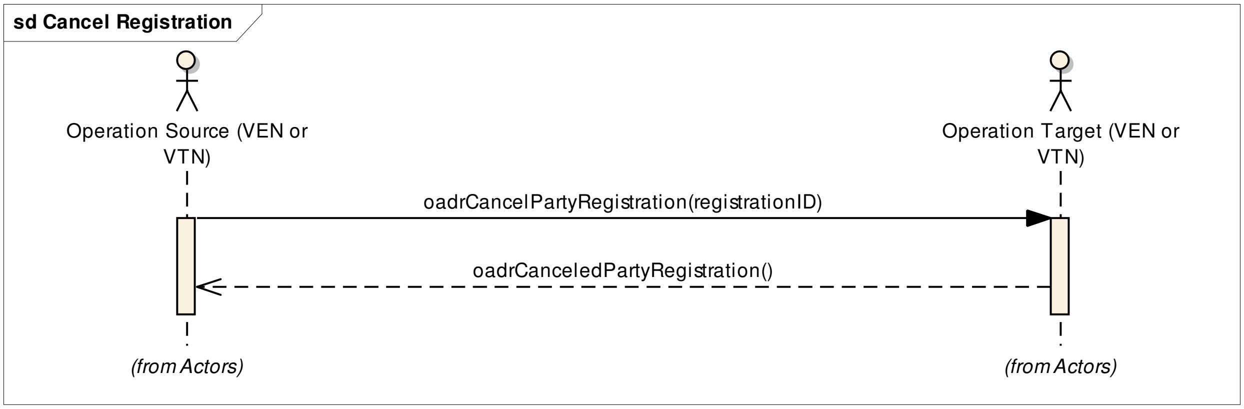

sd Cancel Registration
Operation Source (VEN or VTN)
Operation Target (VEN or VTN)
oadrCancelPartyRegistration(registrationID)
oadrCanceledPartyRegistration()
(from Actors)
(from Actors)

Figure 16 Interaction diagram: Cancel Registration

The sequence diagrams shown are for a PUSH interaction pattern. In an HTTP PULL model, the VEN must periodically poll the VTN using oadrPoll to provide the VTN an opportunity to cancel the registration or request the VEN to reregister. Please refer to the documentation for oadrPoll in section 8.6 for sample sequence diagrams of the PULL interaction pattern.

8.4.2 Registration Information

The following tables outline registration elements defined and whether the information appears in the VEN oadrCreatePartyRegistration request or the VTN oadrCreatedPartyRegistration response.

8.4.2.1 VEN Information in oadrCreatePartyRegistration Payload

Table 3 oadrCreatePartyRegistration payload

Registration Item	Comments
requestID	
registrationID	RegistrationID is only included in this payload when re-registering.
venID	Unless preconfigured, the venID is not included in the initial registration payload, but must be present and correct for re-registration.
oadrProfileName	The profile the VEN selects to use for communication with the VTN. Either 2.0a or 2.0b.
oadrTransportName	The transport the VEN selects to use for communication with the VTN. Either XMPP or simpleHTTP.
oadrTransportAddress	If the selected transport is simpleHTTP or XMPP and the oadrHttpPullModel is false, this must contain the address of the VEN server.
oadrReportOnly	This boolean element is used to indicate the implementation is a Report Only instance of the B profile.
oadrXmlSignature	This boolean element is used to indicate the implementation supports XmlSignatures.
oadrVenName	This is an optional human readable name for the VEN.
oadrHttpPullModel	This boolean element is used to indicate the implementation uses the simpleHTTP PULL exchange model.

8.4.2.2 VTN Information in oadrCreatedPartyRegistration Payload

Table 4 oadrCreatedPartyRegistration payload

Registration Item	Comments
EiResponse Object	Contains standard payload response status information.
RegistrationID	A value used to identify the registration instance. Used for re-registration and cancelation.
venID	The venID is assigned by the VTN and returned in the registration response.
vtnID	Provided to the VEN for inclusion in payloads.
oadrProfiles:oadrProfile:oadrProfileName	A collection of one or more profiles and the associated profile name.
Nested within each profile... oadrTransports:oadrTransport:oadrTransportName	A collection of one or more transports and the associated transport name.
oadrRequestedOadrPollFreq:duration	The polling interval (i.e., the duration between two polls) using oadrPoll should be as requested by the VTN during registration in oadrRequestedOadrPollFreq. Conformance rule 500 defines the constraints for oadrRequestedOadrPollFreq.
oadrServiceSpecificInfo:oadrService:oadrServiceName oadrServiceSpecificInfo:oadrService:oadrInfo:oadrKey oadrServiceSpecificInfo:oadrService:oadrInfo:oadrValue	An object that can contain key - value pair information specific to a particular named service.
oadrExtensions:oadrExtension:oadrExtensionName oadrExtensions:oadrExtension:oadrInfo:oadrKey oadrExtensions:oadrExtension:oadrInfo:oadrValue	An object that can contain key - value pair information specific to a particular named OpenADR extension.

8.5 OpenADR 2.0b EiOpt Service

8.5.1 Service Operations

The OpenADR 2.0 B profile specifies the EiOpt service to create and communicate Opt-In and Opt-Out schedules from the VEN to the VTN. These schedules define temporary changes in the availability, and may be combined with longer term availability schedules and the Market Context requirements to give a complete picture of the willingness of the VEN to respond to EiEvents received by the VEN. The service operations listed in Table 5 are supported:

Table 5 EiOpt payloads

Request Payload	Response Payload	Requestor	Responder
oadrCreateOpt	oadrCreatedOpt	VEN	VTN
oadrCancelOpt	oadrCanceledOpt	VEN	VTN

The VEN uses the oadrCreateOpt payload (see Figure 17) to accomplish one of the following objectives:

- To communicate to the VTN a period of temporary availability for a specific set of eiTargets
- To communicate to the VTN a period of temporary un-availability for a specific set of eiTargets
- To optIn or optOut of a previously acknowledged event for a specific set of eiTargets

oadrCreateOpt payload includes an optID, which can be used in subsequence operations to reference this schedule. The VTNs response to oadrCreateOpt is an oadrCreatedOpt payload that includes an optID, which can be used in subsequence operations to reference this schedule.

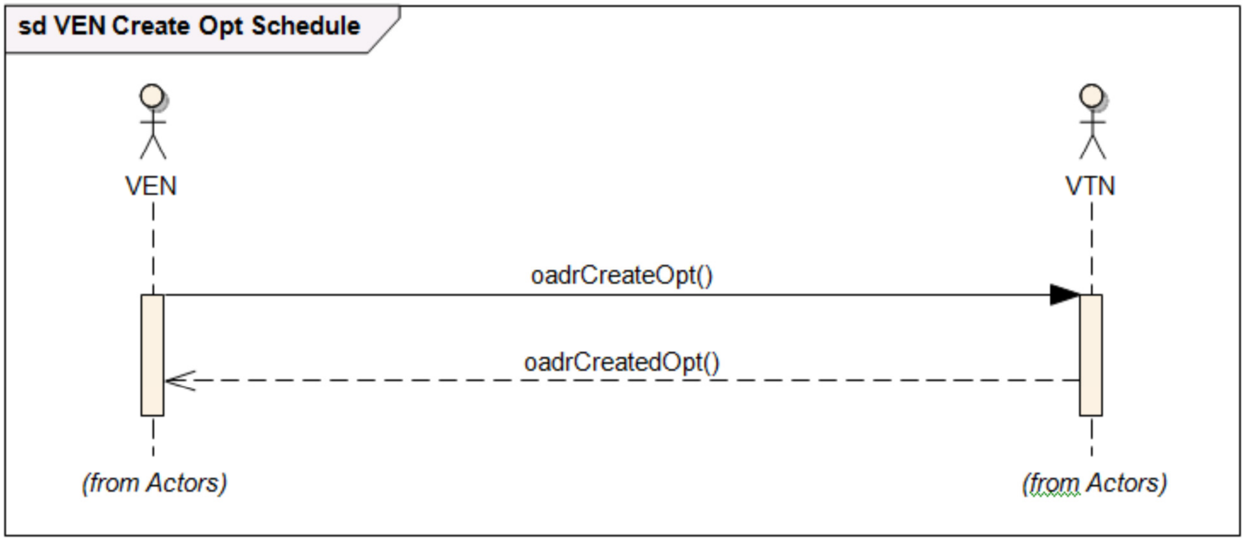


Figure 17 Interaction diagram: Create Opt

The VEN may at any time cancel a temporary availability schedule by using the oadrCancelOpt with an optID referencing the schedule to be canceled (see Figure 18).

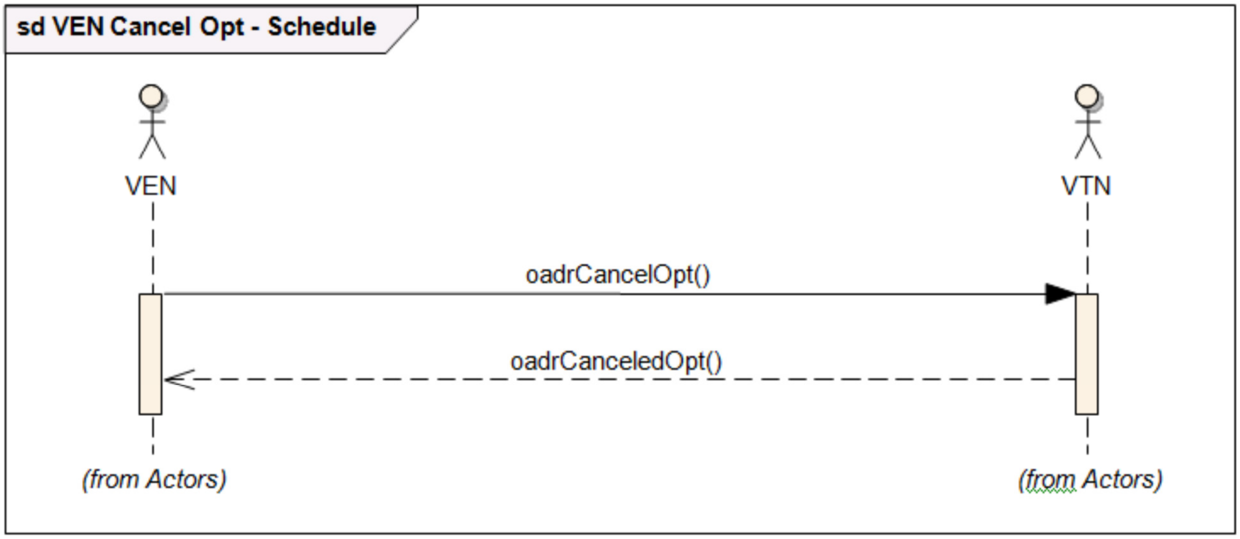


Figure 18 Interaction diagram: Cancel Opt

8.5.2 Detail Requirements

If marketContext is present in the oadrCreateOpt payload, the opt schedule must only apply to the VEN's availability with respect to events generated within the specified marketContext.

A VEN must respond with an EiEvent oadrCreatedEvent payload in response to an oadrDistributeEvent. However, further qualifications of the optType state for the event can be accomplished by sending either an oadrCreatedEvent or an EiOpt oadrCreateOpt payload with the eventID specified. Once oadrCreateOpt is used, it will take precedence and subsequent changes to the opt state by oadrCreatedEvent will be ignored.

The oadrCreateOpt eiTarget must contain the venID of the VEN initiating the opt schedule. If only the venID is specified in eiTarget, then the opt schedule applies to all of the resources associated with that VEN. If additional eiTarget sub elements are defined in addition to venID, these sub elements must be OR'd together to define a subset of the VENs resources that the opt schedule must apply. Note that the VEN may send multiple opt schedules for different sets of resources identified by the eiTarget element.

A new opt schedule (optIn or optOut) sent via oadrCreateOpt must be handled as follows when a previously sent opt schedule is still active:

- If only the venID is specified in eiTarget, the previous opt schedule with respect to future availability is replaced in its entirety by the new opt schedule for all resources associated with the VEN.
- If sub elements are specified in eiTarget in addition to the venID, then an opt schedule must be generated for these resources, overwriting as necessarily any previously defined schedules for specific resources.
- Previously defined opt schedules for resources that are not specified by eiTarget must remain unchanged.

Many aspects of an opt schedule are program specific, including:

- The type of schedules that a VEN is allowed to issue: optIn, optOut, or both
- The default opt state (optIn or optOut) for time periods not defined by either an opt or Avail schedule (if defined)

- Whether optIn or optOut schedules have precedence in the situation where they have overlapping time frames
- Whether overlapping time frames are allowed within the same schedule (optIn or optOut) and if they are not allowed, the expected exception handling behavior
- Whether the VTN is allowed to send events during an optOut period

8.6 OpenADR Poll

As new services are added to the OpenADR profiles, there becomes a need for pull-only VENs to periodically poll the VTN, particularly for VTN-initiated use cases where the information is not necessarily periodic and the VEN cannot predict when a VTN might want to send some information. In the PUSH case this is a non-issue. However in the PULL case, this requires the VEN to periodically poll the VTN, possibly at multiple different service endpoints, to retrieve all the information that the VTN might want to provide. oadrPoll provides a solution which allows the PULL VEN to emulate the PUSH message exchange pattern from VTN to VEN.

oadrPoll is a service independent polling mechanism used by VENs in a PULL model to request pending service operations from the VTN. The VEN queries the poll endpoint and the VTN responds with the same message that it would have "pushed" had it been a PUSH VEN. If there are multiple messages pending a "push," the VEN will continue to query the poll endpoint until there are no new messages and the VTN responds with an eiResponse payload.

The rules for which payloads are valid and how those payloads are delivered are the same as if the VTN had initiated the operations and pushed the payloads to the VEN. Only one operation payload may be sent by the VTN in response to the oadrPoll message.

If a logical response is required by the VTN to the received operational payload, the VEN must send that logical response asynchronously via a transport request. The VTN should acknowledge this logical response with an oadrResponse payload.

The VTN can optionally ignore an oadrPoll if it has not received an expected logical response to a payload delivered as a response to a previous poll. The VTN should be coded such that after some timeout it gives up waiting for the expected response and resumes responding to oadrPoll. The following table outlines the possible response payloads when a VEN uses oadrPoll.

Request Payload	Response Payload	Requestor	Responder
oadrPoll	One of the following: • oadrResponse • oadrDistributeEvent • oadrCreateReport • oadrRegisterReport • oadrCancelReport • oadrUpdateReport • oadrCancelPartyRegistration • oadrRequestReregistration	VEN	VTN

The oadrPoll request has a single sub-element, the venID initiating the oadrPoll. Below is an example payload:

```
<oadr:oadrPoll ei:schemaVersion="2.0b">
        <ei:venID>VEN_123</ei:venID>
</oadr:oadrPoll>
```

The following sequence diagrams illustrate typical patterns using oadrPoll:

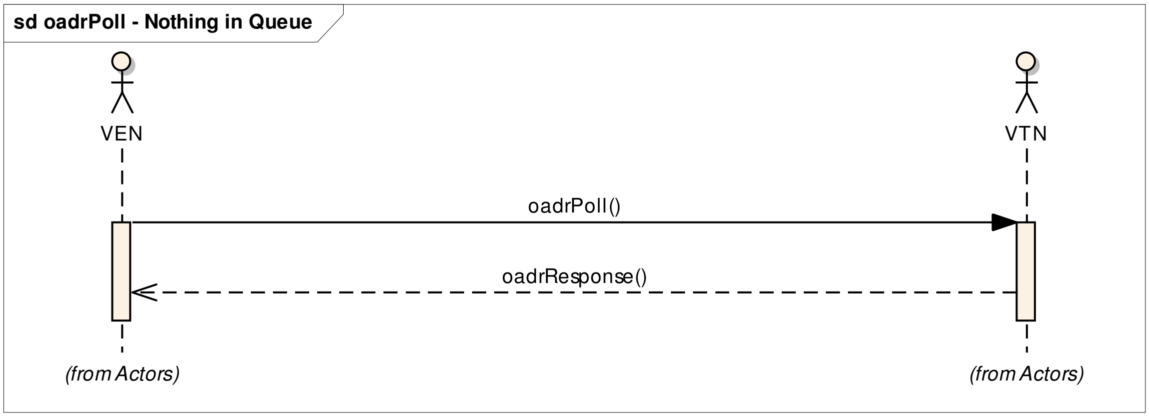


Figure 19 Interaction diagram: oadrPoll (nothing in queue)

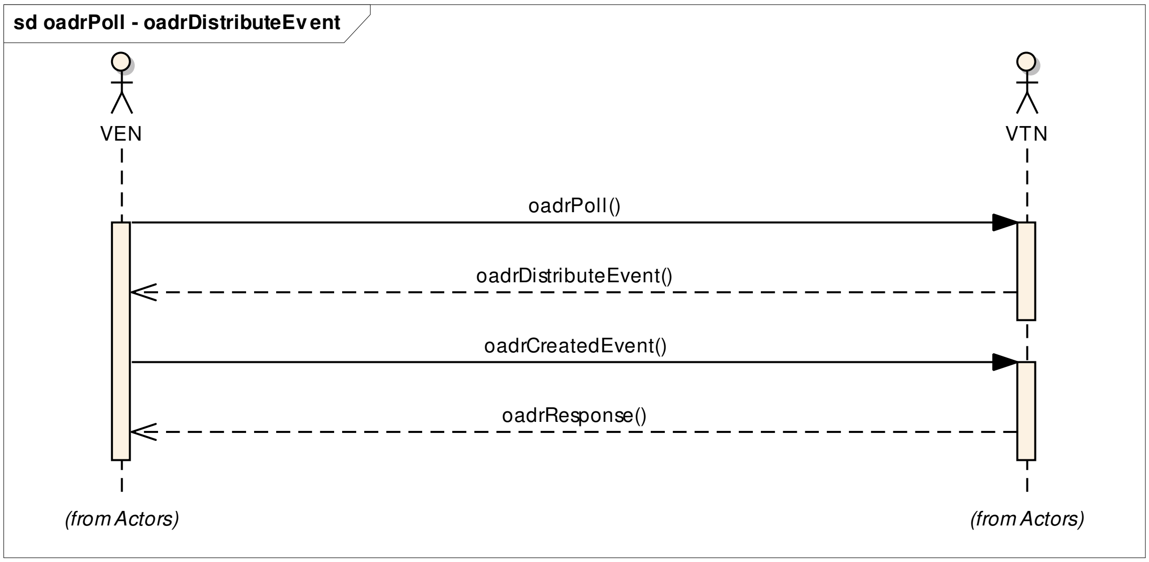


Figure 20 Interaction diagram: oadrPoll (oadrDistributeEvent reply)

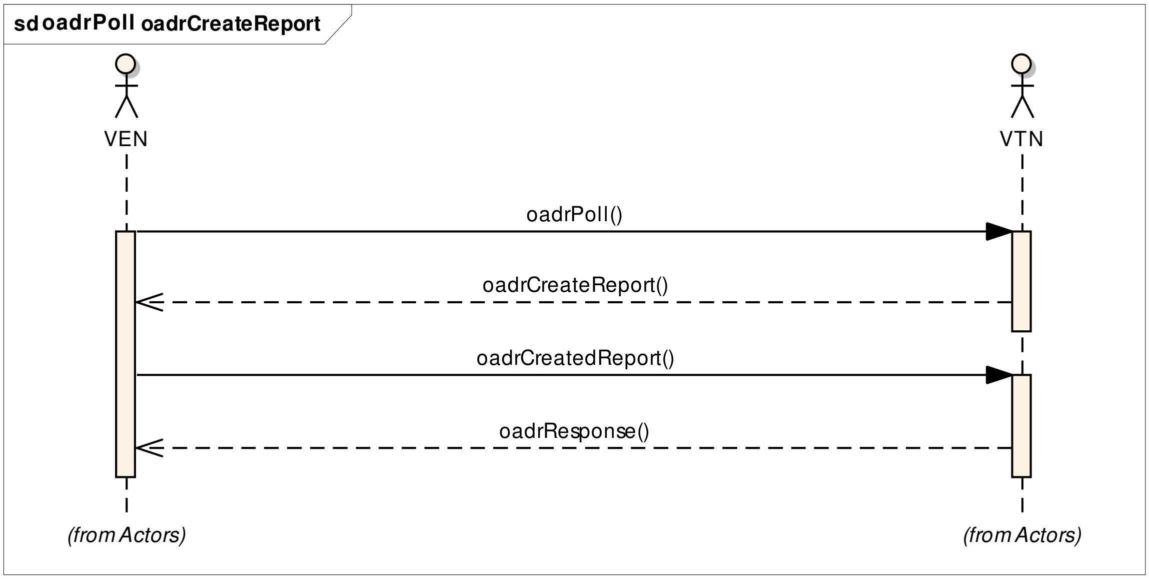


Figure 21 Interaction diagram: oadrPoll (oadrCreateReport reply)

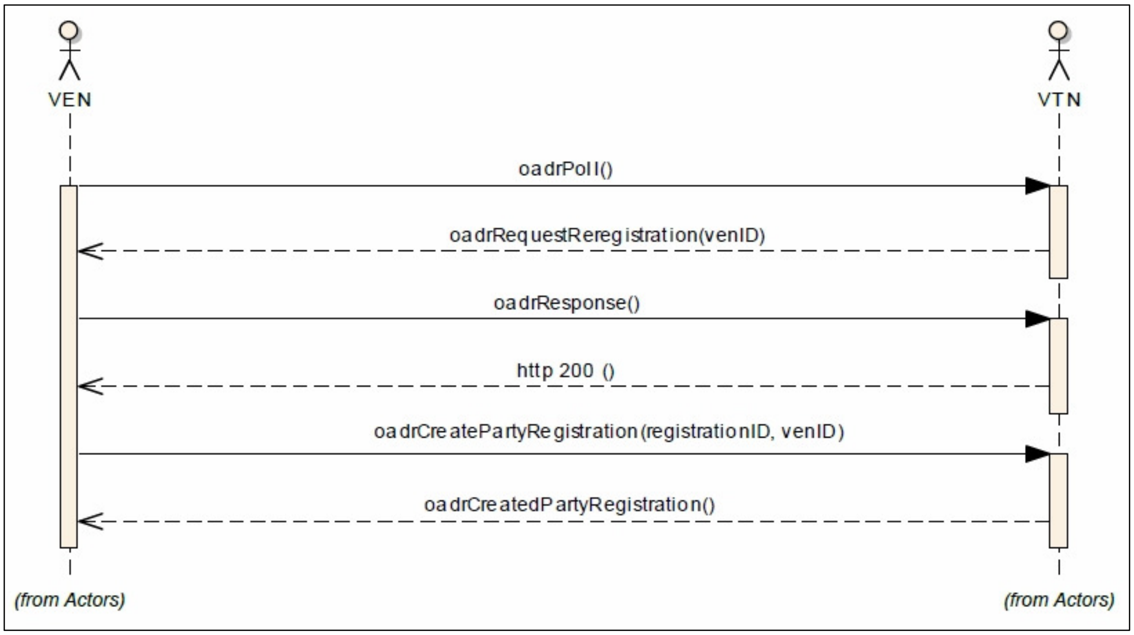

VEN
VTN
oadrPoll()
oadrRequestReregistration(venID)
oadrResponse()
http 200 ()
oadrCreatePartyRegistration(registrationID, venID)
oadrCreatedPartyRegistration()
(from Actors)
(from Actors)

Figure 22 Interaction diagram: oadrPoll (request reregistration reply)

8.7 Application Error Codes

OpenADR 2.0 uses a 3 digit numeric error code starting with the digit “4” to communicate application layer errors in the responseCode element of the various response objects in the schema. There are two general categories of errors: compliance errors and deployment errors. The second digit of the error code must be used to differentiate between these two general categories with “5” indicating compliance errors and “6” indicating deployment errors. Error scenarios documented in the conformance rules will specify which of the error codes listed below should be used. For other error scenarios the VEN or VTN implementer will have to use their best judgment to determine the appropriate application layer error code to use.

Compliance Error Codes
450 - Out of sequence (event ordering, uid ordering, modification number sequencing)
451 - Not Allowed (changing an event in the past)
452 - Invalid ID (eventID, optID, requestID, registrationID, etc.)
453 - Not recognized (reportName, signalName, etc.)
454 - Invalid Data (out of range signal or report data)
455 - Open
456 - Open
457 - Open
458 - Open
459 - Compliance Error - Other

Deployment Errors
460 - Signal not supported (recognized, but not supported)
461 - Report not supported (recognized, but not supported)
462 - Target mismatch (cannot resolve target/market context to VEN or its resources)
463 - Not Registered/Authorized
464 - Open
465 - Open
466 - Open
467 - Open
468 - Open
469 - Deployment Error - Other

9 Transport Protocol

OpenADR 2.0 supports a small number of transport protocols to accommodate different deployment scenarios. 2.0b VENs can either support HTTP or XMPP, or may support both. VTNs must support both HTTP and XMPP.

9.1 Simple HTTP

Simple HTTP in OpenADR 2.0 (a/b) refers to an HTTP implementation that uses HTTP POST over TLS to propagate OpenADR payloads.

9.1.1 PUSH and PULL implementation

9.1.1.1 PUSH Definition

In PUSH mode, 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 oadrDistributeEvent. While this is the most efficient way to execute OpenADR over HTTP, it presents technical difficulties as the VEN may reside behind a network firewall.

9.1.1.2 PULL Definition

In PULL mode, 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 mode has its own limitations, namely latency (due to limited polling frequency) and increased bandwidth requirements.

The PULL mode may involve a ‘two-phase’ execution to complete some operations. This is due to the nature of the VEN initiating the HTTP request. In a PUSH mode, the VTN may notify a VEN of a new event via the oadrDistributeEvent operation. The VTN would send a request with an oadrDistributeEvent payload, to which the VEN responds with HTTP 200 followed by an asynchronous oadrCreatedEvent

However in the PULL model, the VEN requests events from the VTN using oadrPoll, to which an oadrDistributeEvent payload 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 oadrCreatedEvent operation on the VEN.

9.1.2 Service Endpoint URIs

The endpoint names will be of the form:

https://<hostname>(:port)/(prefix/)OpenADR2/Simple/2.0b/<service>

- “prefix” is an optional URI path prefix that may be used to separate OpenADR services from other services that may reside on the same HTTP server.
- “Simple” indicates the simple XML over HTTP protocol
- <service> is the name of the EI service (e.g., “EiEvent”, “EiReport”, “EiOpt”, “EiRegisterParty”, “OadrPoll”).

The “operation” portion of a service is defined by the XML payload sent in a request. E.g., an oadrDistributeEvent payload root element specifies the oadrDistributeEvent operation.

VTN to VEN traffic via HTTP should flow through the same well-defined service endpoints used when communicating in the opposite direction. The oadrCreatePartyRegistraion:oadrTransportAddress element should contain the base address for the VEN, such as "http://myaddress:8080/prefix/". When sending traffic to the VEN, the VTN should concatenate the base address to the well known service end points to form a complete endpoint address such as http://129.6.252.49:8080/prefix/OpenADR2/Simple/2.0b/EiReport.

Note that for implementations that expose both a VEN and VTN (such as an aggregator,) these implementations must use different URI endpoints for their VTN and VEN interfaces. For example, https://mycompany.com/myVTN/OpenADR2/Simple/2.0b/EiEvent and https://mycompany.com/myVEN/OpenADR2/Simple/2.0b/EiEvent.

9.1.3 HTTP Methods

All messages will be sent using the HTTP POST method. This helps to avoid caching and allows all operations to contain a payload in the HTTP request body.

9.1.4 Failure Conditions

The following failures can occur for a given operation:

- TCP (or below) fails
- TLS negotiation 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.

In the case of TCP failure, it is always recommended to retry the operation.
In the case of an HTTP failure, the behavior must be according to HTTP status codes defined in section 9.1.5.
Application-level errors are defined in section 8.7.Timeout failure handling is defined below.

9.1.5 HTTP Response Codes

The following HTTP status codes, defined in [RFC2616], are used in OpenADR:

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:

404 Not Found – the VEN or VTN does not support the requested operation. The requestor must 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 must not re-send the request without first modifying it.

501 Not Implemented – if any request is made with an unsupported HTTP method. The requestor must 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 must indicate to the requestor that it must execute quiesce logic in order to not put further strain on the server. The requestor must 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 oadrResponse payload.

9.1.6 Message Timeouts

There are no prescribed connect or response timeout thresholds for OpenADR 2.0. In general, implementations must allow configurable timeouts in order to handle IP networks with different latency characteristics. HTTP clients must use a request timeout of at least 5 seconds.

9.1.7 Message Retry / Quiesce Behavior

When a request fails for any reason (either due to physical or network-level failure or a timeout) the requestor must institute 'back-off' or quiesce logic to avoid flooding the network or receiver with requests.
Clients must 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. Then the device may retry the request. If subsequent retries fail, quiesce interval must double from the prior interval, again adding a random jitter of plus or minus 10% of that interval. This doubling for subsequent failures must 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 back off algorithm."

9.1.8 PULL Timing

The ability to configure the poll interval for a PULL mode is not defined in OpenADR 2.0a, however VEN implementations must allow the poll interval to be configurable at a millisecond resolution.

In 2.0b, a VTN may configure the minimum poll interval for the VEN by setting oadrCreatedPartyRegistration:oadrRequestedOadrPollFreq accordingly.

VENs must also be configurable to induce some 'jitter' – a random offset from the absolute poll interval – in order to avoid request spikes on the VTN caused by many VENs initiating a PULL request at the exact same instant within the poll interval.

9.1.9 HTTP Headers

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

9.1.9.1 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 must always be "application/xml".

9.1.9.2 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 must honor it and compress the response content using one of the methods given in the request header.

9.1.9.3 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 must 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.

9.1.9.4 Content-Length

The content-length header must be used according to [RFC2616] to indicate content body size of all request and response payloads.
Special Note: 'chunked' transfer encoding (where content-length is unknown) must not be used in OpenADR 2.0. Implementers must assume that the total content body length is known when the response headers are sent, and must not send chunked responses.

9.1.9.5 Content-Type

Must be used for request and response messages, indicating payload MIME type. The appropriate value is "application/xml". The content-type may also specify a character encoding. For OpenADR 2.0, the only supported character encoding is UTF-8. If a charset is included, the entire header value must appear as "application/xml; charset=utf-8".

9.1.9.6 Host

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

9.1.9.7 User-Agent

The requestor may include the User-Agent header but its presence must not be relied upon, nor must it materially affect the behavior of the server handling the request.

9.2 Transport Specific Security

TLS must be used to encrypt all traffic, regardless of the authentication method used. The client must always validate the server's TLS certificate given during the handshake.

9.2.1.1 SSL/ TLS Client Certificate

Client certificates must be used for HTTP client authentication. The entity initiating the request (the client) must have an X.509 certificate that is validated by the server during the TLS handshake. If no client certificate is supplied, or if the certificate is not valid (e.g., it is not signed by a trusted CA, or it is expired) the server must terminate the connection during the TLS handshake.

If the certificate appears valid during the TLS handshake, the connection is established and the HTTP request proceeds. Once the server receives the HTTP request, it must perform authentication, given the credentials in the client certificate. The VTN should use the certificate public key as the primary identification mechanism. The client credentials must be compared to ensure they match the venID that appears in OpenADR payload of the client request. If credentials do not match, or if the server otherwise determines that the request is not authorized, it must respond with an application layer 463 error in the eiResponse object of an appropriate message (e.g., in an oadrCreatedPartyRegistration response if the request from the VEN was an oadrCreatePartyRegistration).

OpenADR Security Certificates are regulated by the [OpenADR 2.0 Certificate Policy].

9.3 XMPP

XMPP is a stateful, bi-directional protocol ideal for transmitting messages in XML format. The core protocol is specified in [RFC6120], and addressing is defined in [RFC6122]. Additional information including accepted extensions can be found at http://xmpp.org/.

9.3.1 Exchange Model Implementation

By nature, XMPP is a bi-directional, stateful protocol. As such, any client utilizing XMPP can implement both PUSH and PULL operations seamlessly, with the exception of oadrPoll which is disallowed over XMPP. PULL operations are simply those initiated by the VEN, and PUSH operations are initiated by the VTN.

9.3.2 Service Endpoints

A Jabber Identifier (JID) defines both VTN and VEN service endpoints, which is similar to an email address. The fully qualified JID performs the same function as an endpoint URI in an HTTP implementation. The definition of JIDs is further described in section 9.3.4.1 of this document.

9.3.3 Service Execution

Because XMPP is a message-based protocol, execution of OpenADR services occurs by passing XMPP messages that contain an OpenADR XML payload.

9.3.4 Implementation of XMPP Features for OpenADR

9.3.4.1 JIDs

While the prospect of using of multiple resources per JID opens some interesting possibilities (e.g., exposing each building at a facility as a different resource,) this strategy should be used with some caution, as same mechanism is not supported by other OpenADR transports such as HTTP.

VEN clients should always authenticate with a fully qualified JID (rather than allow the server to assign a random resource ID.)

VTN clients may choose to expose OpenADR operations either as an XMPP service or as a client JID. Although most OpenADR service operations will be ‘pushed’ from VTN to VEN, a VEN may still need to send some requests to the VTN. Rather than use some configuration to define the JID for VTN services, OpenADR services must be discovered at runtime via service discovery (described in section 9.3.4.9.)

VTN to VEN traffic via XMPP must flow through an endpoint defined by a single fully qualified JID. This VEN JID will be communicated to the VTN as part of the oadrCreatePartyRegistration payload in the oadrTransportAddress element. The resource name used for the JID can be arbitrary such as "client". For instance, the contents of the oadrTransportAddress element might be something like "ven@domain/client".

VEN to VEN traffic is disallowed and must be filtered by the XMPP server.

9.3.4.2 Use of the Packet ‘type’ Attribute

All operations must use the ‘set’ IQ type attribute for all OpenADR service operations.

9.3.4.3 Use of Message versus IQ Packets

Any OpenADR service operations that require an application-level response (e.g., an OpenADR response payload) or a transport-level response code must use an IQ packet. The XMPP protocol then mandates a response IQ from the recipient.

Any OpenADR service operations that do not require a response, such as a broadcast operation (or possibly feedback,) may use a message packet to send the OpenADR2 payload. Handling of the packet should be identical to if it were an IQ, except that the recipient sends no response. This is a bandwidth optimization for use cases where a response is not expected, and a response IQ would be unnecessary.

9.3.4.4 Error Handling

9.3.4.4.1 Transport-Level Status Codes

Under normal conditions, an IQ response will have a type='result' attribute. This is analogous to an HTTP 200 response. OpenADR2 service operations that define an empty HTTP 200 response will simply look like an IQ element with no child payload element:

```
<iq type='result'
   to='ven1234@xmpp.myvo.net/client'
   from='vtn.xmpp.somevtn.net'
   id='1' />
```

9.3.4.4.2 Transport-Level Errors

Operations that result in a transport-level error will have a type='error' and a child <error> element that indicates the transport-level status code as described in [RFC6120] section 8.3. For example:

```
<iq type='error'
   to='ven1234@xmpp.myco.net/client'
   from='vtn.xmpp.somevtn.net'
   id='2'>
   <error type='modify'>
      <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
   </error>
</iq>
```

Since XMPP does not use numeric status codes, direct mapping of HTTP status codes is not possible, and logic for handling transport-level errors is outside the scope of this document. Guidelines for handling different types of XMPP errors can be found in [RFC6120] section 8.3.3.

If a client receives an OpenADR payload that does not adhere to the OpenADR schema, a client should respond with a bad-request error as defined in section 8.3.3.1 of [RFC6120].

Note that application-level errors must be handled the same as in HTTP. That is, a normal 'result' XMPP response should be returned with the application-level error details in the OpenADR payload.

9.3.4.5 Presence

VEN clients must use presence packets to notify subscribed entities of online and offline status as described in [RFC6121] section 4. All other status broadcasts from a client should be considered informational.

After a VEN completes session negotiation, it must send an 'available' presence stanza as defined in Section 4.2.1 of [RFC6121]:

```
<presence />
```

If a VEN deliberately terminates its XMPP connection (e.g., due to a graceful shutdown, not an unexpected, sudden connectivity loss,) it must first send an 'unavailable' presence stanza as defined in Section 4.5.1 of [RFC6121]:

```
<presence type='unavailable'/>
```

The VTN should not generally advertise its presence to the VENs. A VTN could subscribe to VENs and receive presence updates, but may cause scalability issues and should be used with caution.

9.3.4.6 XMPP Ping

Support for XMPP ping (as specified in [XEP-0199]) is mandated from the VEN to the XMPP server. The VEN may notice disconnection from XMPP server and, e.g., reestablish connection or send an alarm if one or more pings are not replied within a certain time. XMPP ping intervals and other parameters related to XMPP ping may be set locally, and exchange of these parameters is not required as part of registration.

9.3.4.7 Authentication

All clients must support SSL/TLS and authentication as defined in section 13.8 and 13.9.4 of [RFC6120]. Clients must also implement Simple Authentication and Security Layer SASL EXTERNAL in order to use certificate authentication as defined in [RFC6120].

The username of the VTN/VEN logging in to the XMPP server must match the Common Name (CN) field of the x509 certificate that is used in the authentication process.

9.3.4.8 Rosters

While rosters are a core feature of XMPP and commonly occur during XMPP session initiation, they have no defined role within OpenADR2 and therefore their use is undefined within the scope of OpenADR2.0. A roster request may be performed by a client, however under most circumstances, the server may simply respond with an empty roster. Optimally, a client (particularly VEN clients) should not request a roster from the server for the purposes of OpenADR.

More specifically, VEN clients should not typically add other entities to its roster, as this will result in additional network overhead due to presence broadcasts.

VTN clients might choose to use the roster as a mechanism to track presence of online VEN clients, however this is not strictly a feature or requirement of OpenADR.

9.3.4.9 Service Discovery

XMPP Service Discovery [XEP0030] must be used to define the address for OpenADR services on the VTN. This allows the VTN flexibility in how it implements services, while removing the need for additional configuration on the VEN, which would be outside the scope of OpenADR.

9.3.4.9.1 Discovery of the OpenADR Feature

After a VEN has completed session initiation, it may perform an [XEP0030] ‘info’ query to the bare domain to which the VEN has authenticated. For example, if a VEN has connected as ‘ven1@xmpp.myco.net/client’ then its initial service discovery query would look like the following:

```
<iq type='get'
   from='ven1@xmpp.myco.net/client'
   to='xmpp.somevtn.net'
   id='info1'>
```

```
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

To which the server would respond:

```
<iq type='result'
    from='xmpp.somevtn.net'
    to='ven1@xmpp.myco.net/client'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='http://openadr.org/openadr2'/>
    <feature var='http://jabber.org/protocol/disco'/>
  </query>
</iq>
```

This indicates, in an XMPP-compliant fashion that the XMPP server supports the OpenADR2 protocol.

9.3.4.9.2 Discovery of OpenADR Service Endpoints

Then VEN may then perform an ‘items’ query, with the ‘node’ set to the OpenADR2 namespace 'http://openadr.org/openadr2#services' as such:

```
<iq type='get'
    from='ven1@xmpp.myco.net/client'
    to='xmpp.somevtn.net'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
        node='http://openadr.org/openadr2#services'/>
</iq>
```

The VTN must then respond with the JIDs used for each OpenADR2 service, for example:

```
<iq type='result'
    from='xmpp.somevtn.net'
    to='ven1@xmpp.myco.net/client'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
        node='http://openadr.org/OpenADR2#services'>
    <item jid='event.openadr2.xmpp.somevtn.net'
         node='http://openadr.org/OpenADR2/EiEvent' />
    <item jid='feedback.openadr2.xmpp.somevtn.net'
         node='http://openadr.org/OpenADR2/EiFeedback' />
  </query>
</iq>
```

Note that all services could use the same JID or different JIDs and JIDs may be subdomains (which are common for XMPP services) or fully-qualified with a ‘localpart’ and resource.

9.3.4.9.3 Service Discovery on VEN Clients

VEN clients may choose to implement service discovery to provide supplemental information about their client (such as client IP address, available disk space, etc.) However such supplemental information is outside the scope of OpenADR. At the bare minimum, if a VEN supports XEP-0030 queries, it must indicate support for the 'http://openadr.org/openadr2' feature, as described in the “Discovery of the OpenADR Feature” section above.

9.3.5 Security Considerations

Beyond authentication (as defined above) additional steps are required to secure an XMPP network. In particular, VEN clients must not be allowed to communicate with each other (at least, not within the scope of OpenADR). Clients may opt to implement some sort of whitelist to control access from different JIDs, which would be similar to implementing a firewall in an HTTP PUSH scenario.

However, ultimate responsibility for controlling access between XMPP clients must be held at the XMPP server level. In most cases, the XMPP server will be logically deployed alongside or as an integrated part of a VTN. The XMPP server can then implement access controls to keep VENs logically isolated so they can only communicate with the VTN.

10 OpenADR 2.0 Security

OpenADR 2.0 aspires to conform to NIST Cyber Security requirements and to follow the guidelines provided by the “Security Profile for OpenADR” prepared by the UCAIug OpenADR Task Force and SG Security Joint Task Force. Furthermore additional NIST guidelines were taken into account to determine the appropriate security for OpenADR profiles. This background exercise was intended to make sure that the OpenADR meets the NIST Cyber Security and as well as any DR service provider deployment requirements. The OpenADR Alliance provides this basic security framework for testing to meet these stringent requirements and understands that the final security scheme would be determined by the DR program deployments.

It is expected that many OpenADR 2.0 implementations will make use of existing cloud computing services and platforms. The OpenADR Alliance has therefore decided to allow currently available security protocols and cipher suites until they are deprecated for the intended use. All certified OpenADR 2.0 products must be upgradable. The mechanism of the upgradability is left to the manufacturers’ implementation.

OpenADR 2.0 uses common security mechanisms, e.g., TLS, without any modification.

Manufacturers shall refer to the latest version of the NIST Special Publication 800-131A when choosing their security algorithm.

The Demand Response Program operator (e.g., utility) will need to require the appropriate level of security for their system

10.1 Architecture and Certificate Types

To provide security services like authentication, confidentiality and integrity, VENs and VTNs must use Public Key Infrastructure (PKI) certificates. Two levels of security are defined for OpenADR 2.0, called ‘Standard’ and ‘High’. The ‘Standard’ security uses TLS for establishing secure channels between a VTN and a VEN for communication. ‘High’ security additionally uses XML signatures providing non-repudiation for documentation purposes (e.g., a signed OpenADR event may be stored in a database for later documentation that an event has actually been received).

OpenADR 2.0 adopts an open architecture for security and will not restrict itself to some specific or proprietary technologies. There are primarily two public key cryptography algorithm options for using PKI certificates, namely RSA and ECC. While RSA is more widely acceptable, ECC provides the benefits of more efficient cryptographic operations like encryption and digital signatures. For the same cryptographic strength, ECC key sizes are much shorter as compared to RSA keys. This is especially important for embedded devices that favor efficient cryptography. As mentioned, RSA is more widely accepted on the Internet and has multiple certificate providers making it easier to select a provider which best matches the requirements.

To retain the benefits of both options and for the purposes of interoperability, the VTNs must support both, ECC and RSA certificates, each from a well-known Certificate Authority (CA) – governed by the [OpenADR 2.0 Certificate Policy]. The final design may have more than two providers but it will be ensured that at least one ECC and one RSA certificate authorities are included. A VEN can choose to use one or more PKI certificates on the device. The only restriction is that a VEN must implement at least one certificate from the approved list of certificate and CAs for the VTN. This list will be decided upon in advance and will be publicly available. To establish a secure communication channel between the VTNs and VENs, VTNs will have to support all the certificates in the approved list of certificates types and CAs. Upon initiating the communication, a VEN will have the choice to use any one of them.

The following parameters will be used for the certificates:

- **ECC** – 256 bits or longer keys.
- **RSA** – 2048 bits or longer keys.
- **Certificate types** – X.509v3

See references.[5]

10.2 Certificate Authorities

The [OpenADR 2.0 Certificate Policy] and the OpenADR/NetworkFX partnership govern the OpenADR Security Certificates.

10.3 Certificate Revocation

Please refer to the [OpenADR 2.0 Certificate Policy] for details.

10.4 TLS and Cipher Suites

OpenADR 2.0 manufacturers shall review [NIST SP 800-131] for deprecation dates and current security requirements. OpenADR 2.0b requires TLS1.2 and the corresponding cipher suites.

Requirements (mandatory for interoperability):

Transport Layer Security: TLS 1.2

Cipher Suites:

ECC – TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

RSA – TLS_RSA_WITH_AES_128_CBC_SHA256

Note that a VTN or VEN may be configured to support any TLS version and cipher suite combination based on the needs of a specific deployment. However in the absence of changes to the default configuration of the VTN or VEN, the behavior of the devices must be as noted above.

Expired certificate must not be used. To avoid service outage caused by expired certificates, deployed VTN and VEN implementations should have appropriate processes or warning mechanisms in place to identify pending security certificate expiration.

10.5 System Registration Process

Registration is defined as the VTN becoming “aware” of the VEN, e.g., exchange of credential and possibly other information so that the VTN can properly authenticate the VEN. Note that this specifically does not include enrolling in DR programs. The OpenADR 2.0b profile includes a reg-

[5] References

1. NIST Special Publication 800-131A.
2. X.509 public key certificates version 3, http://tools.ietf.org/html/draft-igoe-secsh-x509v3-07
3. http://www.certicom.com/images/pdfs/ecc/ds-ecc-cu-102210.pdf
4. http://www.entrust.net/ecc-certs/index.htm

istration service, although specific deployments may elect to do some portions of the registration process out of band.

10.5.1 Certificate Fingerprints

VENs must facilitate registration by providing a “certificate fingerprint” which can be easily transmitted out-of-band to the VTN. On VTN, there should be one-to-one mapping between venID and certificate fingerprint. The fingerprint may then be used by the VTN to identify a VEN when it first connects to the VTN.

The certificate fingerprint will be generated as follows:

1. Perform a SHA-256 hash on the bytes of the DER-encoded client certificate
2. Take the last 10 bytes (from the 32-byte SHA-256 hash), represented as pairs of hexadecimal digits, separated by the colon (ASCII 58) character.

This fingerprint may be generated by the common ‘openssl’ command line tool from a PEM certificate as follows:

```
$ openssl x509 -in client_cert.pem -fingerprint -sha256
SHA256 Finger-
print=D8:40:F6:2B:9D:6D:91:E4:21:21:64:2B:A5:55:76:GB:9C:6C:6B:00:9B:B5:5E:71:FA:
E4:61:75:9C:EF:A4:D6
```

The last 29 characters of the SHA256 hash (5E:71:FA:E4:61:75:9C:EF:A4:D6) must be used as the “certificate fingerprint.” This fingerprint must be printed or otherwise distributed with the VEN so it can be transmitted out-of-band to the VTN during installation time.

The fingerprint could also be computed in python as follows:

```
import ssl, hashlib

bin_cert = ssl.PEM_cert_to_DER_cert( open('client_cert.pem').read() )
sha_hash = hashlib.sha256(bin_cert).digest()
print ':'.join( '%02X' % ord(c) for c in sha_hash[-10:] )
```

10.6 Implementing XML Signatures for OpenADR 2.0 Message Payloads

10.6.1 Introduction to XML Signature

An XML signature uses public key cryptography to digitally sign portions of an XML document for integrity protection of the signed portions of the XML document. A brief description of XML signing and signature verification is provided here for implementation guidance. The complete XML Signature Specification is available at http://www.w3.org/Signature/.

A fundamental feature of the XML signature is the ability to sign individual portions of the XML tree in the document. This is important when the integrity of certain parts of the XML tree needs to be preserved, while the complete document may still be appended by different entities as required. XML signature can be applied to more than one type of resource like section of XML file, XML encoded data, binary-encoded images, text, or character-encoded files.

XML signatures are of three types – i) Enveloped where the signed element being signed is a parent and the signature element is a child; ii) Enveloping where the signature element is the parent of the element being signed; and iii) Detached where the signature element and the signed element do not have a parent child relationship. Detached signatures can reside in the same XML document as the content being signed, essentially living as sibling element or they can be in a different documenting that the content being signed.

For the “high security” profile in OpenADR, detached signatures with siblings are used.

10.6.2 Components of XML Signatures

Reference URI: This element provides a reference to the resource that is being signed using a URI.

Digest Value: This element contains the digest of the resource that is being signed.

Signature Value: This element contains the resource digest signed by the private key of the signing authority.

Key Info: The key info element contains the information of the key that should be used for signature verification. This contains information on the public key of the signing authority.

10.6.3 Creating XML Signatures

The first step in creating XML signatures is to identify the resource that needs to be signed. For OpenADR 2.0 specification, ‘Detached’ signatures living in the same XML document as sibling element to the data object being signed must be used. The VTN and VEN devices should use the cipher suites that are used for TLS connections. As such there will not be any need for supporting additional cipher suites for creating and verifying message digests. SHA-256 is mandatory for creating message digests and X.509 must be used for public key certificates. Refer to [RFC3275] for further details how to create XML signatures.

An example XML signature is shown in Figure 23.

```
<oadr:oadrPayload>
    <ds:Signature>
        <ds:SignedInfo>
            <ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
            <ds:SignatureMethod
                Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
            <ds:Reference URI="#signedObject">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
                <ds:DigestValue>5H8v6B/z+YMDBR61xKdUZVkCLvUnC/rVfEcrZ5IAFDY=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="#prop">
                <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
                <ds:DigestValue>7H7U/}kg%yHHFDT7jb^*jvR80KHffdR%7hHFVdEttGJU</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>UjBsR09EbGhjZ0dTQUxNQUFBUUNBRU1tQ1p0dU1GUXhEUzhi</ds:SignatureValue>
        <ds:KeyInfo>
            <ds:KeyName>key123</ds:KeyName>
        </ds:KeyInfo>
        <ds:Object Id="prop">
            <ds:SignatureProperties xmlns:dsp="http://openadr.org/oadr-2.0b/2012/07/xmldsig-properties">
                <ds:SignatureProperty>
                    <dsp:ReplayProtect>
                        <dsp:timestamp>2006-05-04T18:13:51.0</timestamp>
                        <dsp:nonce>nonce0</nonce>
                    </ds:ReplayProtect>
                </ds:SignatureProperty>
            </ds:SignatureProperties>
        </ds:Object>
    </ds:Signature>
    <oadr:oadrSignedObject Id="signedObject">
        <oadr:oadrDistributeEvent ei:schemaVersion="2.0b">
            ...
        </oadr:oadrDistributeEvent>
    </oadr:oadrSignedObject>
</oadr:oadrPayload>
```

Figure 23 XML signature example

The different fields of the XML signature are explained below. The <oadrPayload> is the root element for the XML document. It contains two children – <Signature> and <oadrSignedObject>. <Signature> contains the signature element, where as the <oadrSignedObject> contains the OpenADR payload that is being signed.

<Signature> element includes the entire XML signature. The <Reference> element contains the URI pointing to the <oadrSignedObject> element. <DigestMethod> points to the algorithm for creating the digest of the resource being signed. In the example, SHA-256 is used for creating the message digest. <DigestValue> is the alpha-numeric value of the digest. The element <SignedInfo> contains all these information and the Canonicalization Method that was used. Canonicalization is important to have the same verification of XML signatures with different textual representations.
<KeyInfo> element contains the information about the public key of the signing authority that should be used to verify the signature. It contains information including <SubjectName> and the corresponding X.509 certificate of the signing authority that contains the public key.

<Object> element contains a list of <SignatureProperties>. An OpenADR implementation supporting XML signatures must at least insert the <ReplayProtect> element as <SignatureProperty>. The ReplayProtect element must contain the dateTime when the payload is sent to the other party (not when it is created), as well as a random nonce. Note that the ReplayProtect element was part of an earlier version of the W3C “XML Signature Properties” document[6], but has been re-

[6] http://www.w3.org/2008/xmlsec/Drafts/xmldsig-properties/Overview.html

moved because of lack of interoperable implementations. OpenADR uses the same element but in its own namespace.

Digest of the whole <SignedInfo> element is created and signed using the private key of the signing authority.
The allowable algorithms for the signature method are: RSA-SHA256, and ECDSA-SHA256. The allowable algorithm for the digest method is: SHA256. Refer to conformance rule 514 for details.

10.6.4 Verifying XML Signatures

The following steps should be performed to verify an XML signature (refer to [RFC3275])

1. Create a digest of the <SignedInfo> element using the digest mechanism indicated in the <DigestMethod> element (i.e., SHA-256). Verify the <SignatureValue> element using the key pointed out in the <KeyInfo> element. If the verification value is the same as the digest, the first verification step is complete.

2. Calculate the digest of the <oadrSignedObject> field and make sure the calculated digest is the same as the value in the <DigestValue> field.

3. Verify if a <ReplayProtect> element is contained as <SignatureProperty>. Reject the payload if the current date and time on the device differs from the value in the <ReplayProtect> more than a predefined value. In addition, the nonce may be used for further protection against replay attacks.

11 Conformance

11.1 OpenADR 2.0 conformance statement

In order to claim conformance to this profile specification, a VTN, VEN or VTN/VEN combination must conform to all statements made in this document as well as the [OpenADR 2.0 PICS] document. Product variations can be found in the product matrix in Figure 1. Further, a product must be tested at an authorized test service provider and undergo certification managed by the OpenADR Alliance.

11.2 OpenADR 2.0b Profile Conformance Rules

11.2.1 EiEvent – from 2.0a

Note that 2.0b devices must also conform to the A profile conformance rules listed in this section. Some rules have a note "A Profile Only"; these apply only to 2.0b VTNs communicating with 2.0a VENs.

Conformance Rule	Requirement
1	**VEN/VTN, Date-Time and Duration** The date-time values MUST be specified using [ISO8601] utc-time (also called *zulu time*). Example: 2013-04-22T15:26:44.123Z. Fractional seconds in date-time values are allowed by [ISO8601], however it is the responsibility of the receiving device to truncate the fractional seconds if necessary. Although [ISO8601] allows representation of Zulu time with a zero hour offset (e.g, 2013-04-22T15:26:44.123+0000), it MUST NOT be used. Representation of duration also follows [ISO8601]. However, use of decimal values MUST NOT be used in OpenADR.
2	**VTN, EiEvent Service, oadrDistributeEvent Payload** The uid element is REQUIRED for each eiEventSignal interval. Within a single oadrDistributeEvent eiEventSignal, uid MUST be expressed as an interval number with a base of 0 and an increment of 1 for each subsequent interval.
3	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDistributeEvent priority element – This is the priority of this event relative to other events. The lower the number higher the priority. A value of zero (0) indicates no priority, which is the lowest priority by default.
4	**VTN, EiEvent Service, oadrDistributeEvent Payload** A new event MUST start with a modificationNumber of 0.
5	**VTN, EiEvent Service, oadrDistributeEvent Payload** Each modification of the oadrDistributeEvent eiEvent object, excluding createdDateTime, eventStatus, and currentValue, MUST cause the modificationNumber to increment by 1. Exception: An eventStatus change to "cancelled" MUST cause the modification number to increment by 1.

6	**VEN, EiEvent Service, oadrDistributeEvent Payload** The presence of any string except “false” in the oadrDistributeEvent testEvent element MUST be treated as a trigger for a test event.
7 **A Profile Only**	**VTN, EiEvent Service, oadrDistributeEvent Payload** The oadrDistributeEvent eiEvent object MUST contain only one event signal and that signal MUST have a signalName of “SIMPLE” (in either upper or lower case, i.e., “simple”).
8	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDistributeEvent eventSignal interval durations for a given event MUST add up to eiEvent eiActivePeriod duration.
9	**VTN, EiEvent Service, oadrDistributeEvent Payload** oadrDistributeEvent eiEventSignal’s with a signalName of “SIMPLE” MUST use signalPayload values of 0=normal; 1=moderate; 2=high; 3=special.
10	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN MUST change the oadrDistributeEvent eventStatus to “cancelled” when communicating an event cancellation to the VEN. Note that the modificationNumber is incremented by 1 when issuing a cancellation.
12	**VEN, EiEvent Service, oadrCreatedEvent Payload** The VEN MUST respond with an oadrCreatedEvent to an event in oadrDistributeEvent based upon the value in each event’s oadrResponseRequired element as follows: Always – The VEN MUST respond to the event with an oadrCreatedEvent eventResponse. This includes unchanged, new, changed, and canceled events. Never – The VEN MUST NOT respond to the event with a oadrCreatedEvent eventResponse Note that oadrCreatedEvent event responses SHOULD be returned in one message, but MAY be returned in separate messages.
13	**VTN, EiEvent Service, oadrDistributeEvent Payload** EventStatus MUST always transition from “far” to “near” to “active”. The transition to NEAR occurs at the start of the x-eiRampUp period if defined. If x-eiRampUp is not defined the transition will move from “far” to “active” at the dtstart time.
14	**VTN, EiEvent Service, oadrDistributeEvent Payload** If currentValue is included in the payload, it MUST be set to 0 (normal) when the event status is not “active” for the SIMPLE signalName.

15	**VTN, EiEvent Service, oadrDistributeEvent Payload** A list of events sent by VTN in an oadrDistributeEvent payload MUST be ordered as follows: 1. “active” events have priority over pending events. 2. Within events with “active” event status, priority is determined by Priority in the Event Descriptor. 3. Between “active” events with the same priority, the one with the earlier start time has the higher priority. 4. Between pending (i.e., “far” or “near”) events the one with the earlier start time has the higher priority 5. After processing rules 1-4, if Priority is still indeterminate within a set of events AND if the VTN is responding to an oadrRequestEvent with replyLimit set to a value less than the number of pending or “active” events, the VTN MUST maintain a fixed order between successive replies to oadrRequestEvent while the ordering remains indeterminate. Note: A cancellation should be ordered according to whatever rules were applied before the event was canceled.
16	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN MUST recognize the state of the eiCreatedEvent optType element, both optIn and optOut, except when a B profile VEN responds to an event with oadrCreateOpt, in which case conformance rule 206 MUST apply.
17	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN MUST recognize an asynchronous eiCreatedEvent optOut for a previously acknowledged event, except when a B profile VEN had previously responded to an event with oadrCreateOpt, in which case conformance rule 206 MUST apply.

18	**VEN/VTN, EiEvent Service** VTNs may send overlapping events. The behavior of VENs when processing overlapping events is undefined. DR deployments using overlapping events should define the expected behavior and should ensure that VENs participating in the program support the expected behavior. The following rules shall be used when determining if events are overlapping: Event 1 = First event in a sequence Event 2 = Second event in a sequence Start Time = dtStart value in event payload Duration = overall event duration value in payload Randomization Window = startafter value in payload Effective End Time = Event Start Time plus Event Duration plus Randomization Window If Event 2 Start Time is equal to or greater than Event 1 Effective End Time, then Event 2 is NOT considered to be overlapping. If Event 2 Start Time is less than Event 1 Effective End Time, then Event 2 is overlapping. When an event is cancelled the Effective End Time for the event becomes the time the cancellation is acknowledged plus the Randomization Window for the purpose of calculating if a subsequent event is overlapping If events are adjacent, where a subsequent event's Start Time is equal to the current event's Effective End Time, the VEN shall treat the end times as non-inclusive and the start times as inclusive for the purpose of determining which of the two concurrent event signal interval values to use at the instant of transition between the two events.
19	**VEN, EiEvent Service, oadrDistributeEvent Payload** If an oadrDistributeEvent payload has as mix of valid and invalid events, the VEN implementation MUST only respond to the relevant valid events and not reject the entire message.
20	**VTN, EiEvent Service, oadrDistributeEvent Payload** At any time, a VTN MAY change any element or attribute of a pending (i.e., “far” or “near”) or “active” event as long as it does not pertain to the past.
21	**VEN/VTN** If venID, vtnID, or eventID value is included in the payload, the receiving entity MUST validate that the ID value is as expected and generate an error if an unexpected value is received. Exception: A VEN MUST NOT generate an error upon receipt of a canceled event whose eventID is not previously known.
22	**VEN, EiEvent Service, oadrDistributeEvent Payload** If no sub elements are present in oadrDistributeEvent eiTarget, the presumption is that the recipient is the intended target of the event. If multiple criteria are present in eiTarget sub-elements, the values are OR’d together to determine whether the VEN is a target for the event. However, the VENs behavior with respect to responding to an event when it matches one of the eiTarget criteria is implementation dependent.

23	**VEN/VTN, EiEvent Service, oadrRequestEvent Payload** oadrRequestEvent MUST only be sent in VEN to VTN direction.

25	**VTN, EiEvent Service, oadrCreatedEvent Payload** The following rules MUST be followed with respect to application level responses for multiple events in an oadrCreatedEvent: 1) If the oadrCreatedEvent:eiResponse indicates failure, there is no need to examine each element in the eventResponses (eventResponses is not required to be part of the oadrCreatedEvent payload in an error condition, as specified in rule 35). 2) If the oadrCreatedEvent:eiResponse indicates success, the VTN SHOULD evaluate each eventResponse:responseCode to discover which optType state and response codes the VEN recorded for each event.
27	**VTN, EiEvent Service, oadrRequestEvent Payload** If a value is provided in the oadrRequestEvent replyLimit element, the VTN MUST only return the number of events specified by the value in its oadrDistributeEvent response.
29	**VTN, EiEvent Service, oadrDistributeEvent Payload** The oadrDistributeEvent currentValue, if included in the payload, for each eiEvent eiEventSignal MUST accurately reflect the signalPayload value for the active interval in an executing event.
30	**VEN, EiEvent Service, oadrDistributeEvent Payload** The VEN MUST randomize the dtstart time of the event if a value is present in the startafter element. The randomized dtStatrt time should be between the requested dtstart and dtstart plus startafter. Event completion times are determined by adding the event duration to the randomized dtstart time. Modifications to an event SHOULD maintain the same random offset, unless the startafter element itself is modified.
31	**VEN, EiEvent Service, oadrDistributeEvent Payload** The VEN MUST recognize and act upon values specified in the subelements of activePeriod including: • dtstart • duration • tolerance • x-eiRampUp (positive and negative) • x-eiRecovery (positive and negative) Note: x-eiRampup and x-eiRecovery are not testable requirements.
32	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** The VEN MUST recognize and act upon values specified in the subelements of intervals including: • duration • signalPayload

33	**VEN/VTN** The implementation MUST provide an application layer error indication as a result of the following conditions: • Missing expected information (459) • Payload not of expected type (459) • ID not as expected (452) • Illogical request – Old date on new event, durations do not add up correctly, etc. Note that a schema validation error will most likely cause a transport rather than application layer error indication.
35	**VEN, EiEvent Service, oadrCreatedEvent Payload** The eventResponses element in oadrCreatedEvent is REQUIRED, except when an error condition is reported in eiResponse.
36	**VEN, EiEvent Service, oadrCreatedEvent Payload** An event cancellation received by the VEN MUST be acknowledged with an oadrCreatedEvent with the optType element set as follows, unless the oadrResponseRequired is set to “never”: optIn = Confirm to cancellation optOut = Cannot cancel Note: Once an event cancellation is acknowledged by the VEN, the event MUST NOT be included in subsequent oadrCreatedEvent payloads unless the VTN includes this event in a subsequent oadrDistributeEvent payload.
37	**VEN** A simple HTTP VEN implementation MUST support the PULL model and MAY optionally also support push.
38	**VTN** A VTN MUST support all transports and exchange models including HTTP PUSH, HTTP PULL, and XMPP PUSH.
40	**VTN, EiEvent Service, oadrDistributeEvent Payload** The optional eiResponse object in oadrDistributeEvent MUST be included when responding to oadrRequestEvent or to an oadrPoll.
41	**VTN, EiEvent Service, oadrDistributeEvent Payload** The VTN MUST send a requestID value as part of the oadrDistributeEvent payload. Note: The requestID value is not required to be unique, and in fact may be the same for all oadrDistributeEvent payloads. That there are two requestID fields in oadrDistributeEvent. The field that must be populated with a requestID is located at oadrDistributeEvent:requestID.

42	**VEN/VTN, EiEvent Service, oadrCreatedEvent Payload** A VEN receiving and event in an oadrDistributeEvent payload MUST use the received requestID value in the eiCreatedEvent:eventResponse when responding to the event an with oadrCreatedEvent payload. This includes any and all subsequent EiCreatedEvent messages that may be sent to change the opt status of the VEN. The eiResponse:requestID in oadrCreatedEvent MUST be left empty if the payload contains eventResponses. The VTN MUST look inside each eventResponse for the relevant requestID.
43	**VEN, EiEvent Service, oadrDistributeEvent Payload** The VEN MUST NOT make any assumptions regarding the uniqueness of requestID values received from the VTN in the oadrDistributePayload.
45	**VEN/VTN** Messages sent between VENs and VTNs MUST NOT include a schemaLocation attribute.
46	**VEN/VTN** Optional elements do not need to be included in outbound payloads, but if they are, the VEN or VTN receiving the payload MUST understand and act upon those optional elements. Empty or omitted elements are considered equal unless a conformance rule explicitly states that element tags must be excluded from the payload.
47	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** An event with an overall duration of 0 indicates an event with no defined end time and will remain active until explicitly canceled.

48	**VEN/VTN** When a VTN or VEN receives schema compliant OpenADR payload that has logical errors, the receiving device MUST provide an application layer error indication of 4xx in the eiResponse element of the payload. The detailed error message number is informational and not a requirement for response to a specific scenario, unless otherwise stated in the conformance rules. If the error is in an event contained in an oadrDistributeEvent payload and there are otherwise no application errors in the oadrDistributeEvent, the oadrCreatedResponse SHOULD set the eiResponse:responseCode to 200 and SHOULD report the status code and opt status for each event in the eventResponse element of oadrCreatedEvent. If no logical error is detected for an event, the status code is set to 200 and the opt status included in the oadrResponse element. The following logical errors MUST be detected by implementations • VEN receives non-matching market context (462) • VEN receives non-matching eiTarget (462) • VEN receives unsupported signalName (460) In case of a logical error for an event, optType is set to optOut. Exception: On- a deployment specific basis, a VEN MAY be configured so that it accepts any marketContext. In this case, "any" valid URI will be considered matching with respect to this requirement When the VTN receives an oadrCreatedEvent with logical errors in one of the eventResponses, it SHOULD set the eiResponse:responseCode in the oadrResponse reply to: • VTN receives non-matching eventID in at least one of the oadrResponses in the oadrCreatedEvent (452) • VTN receives mismatched modificationNumber in at least one of the oadrResponses in the oadrCreatedEvent (450) In case there are multiple errors (e.g., one non-matching eventID and one non-matching modificationNumber), any 4xx error code (e.g., 452) can be used. When the VEN receives the oadrResponse, it could resend oadrCreatedEvent's, one for each event instead several grouped into one oadrCreatedEvent.
50	**VTN, EiEvent Service, oadrDistributeEvent Payload** In both the PUSH and PULL model, oadrDistributeEvent MUST contain all existing events which have the eventStatus element set to either FAR, NEAR, or ACTIVE. Events with an eventStatus of “cancelled” MUST be included in the payload upon change to the modificationNumber, but once the cancellation is acknowledge by the VEN, MUST not be included in subsequent payloads as stated in rule 36. VTN MAY continue to send events that the VEN has opted out of.

51	**VEN/VTN** PULL implementations may only use payloads where the transport layer request is in the VEN to VTN direction. PUSH implemenations, both HTTP and XMPP, may use any payload in a valid exchange sequence except for oardPoll.
52	**VTN, EiEvent Service, oadrDistributeEvent Payload** If a VTN requests acknowledgment of a canceled event with oadrResponseRequired of always, the VTN MUST continue to send the canceled event to the VEN until the event is acknowledged, eventStatus transitions to the complete state, or some well-defined number of retries is attempted.
53	**VEN/VTN** VTNs and VENs that support simple HTTP mode MUST support the following HTTP headers: • Host (Onlyrquired in HTTP request) • Content-Length • Content-Type of application/xml
56	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is not aware then the VEN SHOULD process the event and add it to its list of known events.
57	**VEN/VTN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is already aware of, but with a higher modification number then the VEN should replace the previous event with the new one in its list of known events.
58	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent with an eventID that the VEN is already aware of, but which has a lower modification number than one in which the VEN is already aware then this is an error and the VEN SHOULD respond with the appropriate error code (450). Note that this is true regardless of the event state including “cancelled”.
59	**VEN, EiEvent Service, oadrDistributeEvent Payload** If the VTN sends an oadrEvent object with the eventStatus set to “cancelled” and has an eventID that the VEN is aware of then the VEN SHOULD cancel the existing event and delete it from its list of known events.
60	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** If the VTN sends an oadrEvent object with the eventStatus set to “cancelled” and has an eventID that the VEN is not aware of then the VEN SHOULD ignore the event since it is not currently in its list of known events, but still MUST respond with the createdEvent if required to do so by oadrResponseRequired.
61	**VEN, EiEvent Service, oadrDistributeEvent Payload**

	If the VTN sends the oadrDistributeEvent payload and it does not contain an event for which the VEN is aware (i.e., in its list of known events) then the VEN MUST delete it from its list of known event (i.e., implied cancel). Exception: A VEN that has an active event that cannot be immediately stopped for operational reasons MAY leave the event in its data store until the event expires or the event can be stopped.
62	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** The VEN MUST process every oadrEvent event message (new, modified, canceled, etc.) that it receives from the VTN in an oadrDistributeEvent payload and it MUST reply with a createdEvent message for every EiEvent message in which the responseRequired is set to “always”. Furthermore if the responseRequired is set to “never”, the VEN MUST NOT respond with a createdEvent message. It is at the complete discretion of the VTN as to whether responses are required from the VEN. Note that this rule is universal and applies to all scenarios including the following: • The event is one in which the VEN is already aware. • The event is being canceled and the VEN did not even know it existed • It does not matter how the EiEvent payloads were delivered, i.e., PUSH, PULL or as the result of being delivered in an ALL payload
63	**VTN** The VTN MUST NOT include more than one venID in the oadrDistributeEvent eiTarget.
64 **A Profile Only**	**VEN, EiEvent Service** A PULL VEN MUST respond to all received events before initiating another polling cycle.
65	**VEN, EiEvent Service, oadrDistributeEvent, oadrCreatedEvent Payload** When an event containing a randomization value in the startafter element is canceled, either explicitly or implicitly, the VEN MUST randomize its termination of the event. The randomization window SHOULD be between 0 and a duration equal to the value specified in startafter.
66	**VEN/VTN, EiEvent Service, oadrDistributeEvent, Payload** If a VTN sends an oadrDistributeEvent payload containing an event with a startafter element with a value greater than zero, the VTN MUST continue to include the event in oadrDistributeEvent payloads, even if the event is complete, until current time is equal to dtstart plus duration plus startafter. The receipt of an eventStatus equal to completed MUST NOT cause the VEN to change its operational status with respect to executing the event.

67	**VEN/VTN** VTN and VEN MUST support TLS 1.2. The default cipher suite selection MUST be as follows: • The VEN client MUST offer at least one of the default cipher suites listed below. • The VEN server MUST support at least one of the default cipher suites listed below and MUST select one of the default cipher suites regardless of other cipher suites that may be offered by the VTN client. • The VTN client MUST offer both the default cipher suites listed below. • The VTN server MUST support both of the default cipher suites listed below and MUST select one of listed the default cipher suites regardless of other ciphers that may be offered by the VEN client. Default cipher suites for TLS1.2: • TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA256 Note that a VTN or VEN MAY be configured to support any TLS version and cipher suite combination based on the needs of a specific deployment. However in the absence of changes to the default configuration of the VTN or VEN, the behavior of the devices MUST be as noted above.
68	**VEN/VTN** Both VTNs and VENs MUST support client and server X.509v3 certificates. A VTN MUST support both an ECC and RSA certificate. A VEN MUST support either an RSA or ECC certificate and MAY support both. RSA certificates MUST be signed with a minimum key length of 2048 bits. ECC certificates MUST be signed with a minimum key length of 224 bits. ECC Hybrid certificates MUST be signed with a 256 bit key signed with an RSA 2048 bit key.
	VTN/VEN, EiEvent Service, oadrDistributeEvent, Payload The signalType element contained in a SIMPLE signal MAY be any of the supported enumerated values. The value used is informational and provides a hint as to the nature of the relative values specified in the signalPayload.
70	**VTN, EiEvent Service, IDs** VTN MUST ensure that venID is unique in the scope of VTN. Namely, all VENs are assigned unique IDs. VTN MUST ensure that eventID is unique in the scope of VEN. Namely, while VTN can distribute events with the same eventID to multiple VENs, it cannot sent two distinct events to the same VEN with the same eventID.

11.2.2 EiEvent – Additional 2.0b Conformance Rules

Conformance Rule	Requirement
100	**VTN, EiEvent Service, oadrDistributeEvent Payload** For both eiEventSignal and eiEventBaseline the following substitution group items MUST be used: • streamPayloadBase = signalPayload • payloadBase = payloadFloat The number of signalPayload elements in each interval MUST be equal to 1.
101	**VTN, EiEvent Service, oadrDistributeEvent Payload** The uid element is REQUIRED for each eiEventBaseline interval. Within a single eiEventBaseline, uid MUST be expressed as an interval number with a base of 0 and an increment of 1 for each subsequent interval.
102	**VTN, EiEvent Service, oadrDistributeEvent Payload** For both eiEventSignal and eiEventBaseline, the interval duration element must appear in each interval and the sum of interval durations MUST add up to overall duration element specified in eiActivePeriod:properties:duration for event signals and eiEventSignal:eiEventBaseline:duration for baselines.
103	**VTN, EiEvent Service, oadrDistributeEvent Payload** For both eiEventSignal and eiEventBaseline, the dtstart element MUST NOT be included in the interval specification.
104	**VTN, EiEvent Service, oadrDistributeEvent Payload** If the signalName is one of the well-known alliance signals, then signalType, Units, and allowable values MUST be as shown in "Table 1 Signals"
105	**VTN, EiEvent Service, oadrDistributeEvent Payload** The eiNotification element MUST be included as a sub element of activePeriod. Note that this element is REQUIRED in the A profile, but OPTIONAL in the B profile, so this conformance rule simply ensures it is included in B profile events.
106	**VTN, EiEvent Service, oadrDistributeEvent Payload** B profile oadrDistributeEvent eiEvent's MAY contain the "SIMPLE" event signal in addition to other signals, however the relationship between the SIMPLE signal and other signals in the event are deployment specific.

107	**VTN, EiEvent Service, oadrDistributeEvent Payload** If multiple signals are present for an event, each signal MUST have a signalID value that is unique within the scope of the event.
108	**VTN, EiEvent Service, oadrDistributeEvent Payload** If one or more resourceIDs are present as sub-elements of eiEventBaseline, the baseline represents the aggregate of the listed resources. No assumptions should be made regarding the relationship between the eiEvent:eiTarget object and the source of the baseline data.
109	**VEN, EiEvent Service, oadrDistributeEvent Payload** A VEN MUST generate an oadrCreatedEvent application level 460 error if any requested signal in an event cannot be supported. Supported means that the requested combination of signalType and Unit cannot be currently supported.
110	**VTN, EiEvent Service, oadrDistributeEvent payload** The value of currentValue when an event is not active is undefined for event signals other than SIMPLE. See conformance rule 14 for the currentValue requirements for the SIMPLE signal.

111	**VTN/VEN, EiEvent Service, oadrDistributeEvent payload** eiTarget is defined at both the event and signal level of the oadrDistributeEvent payload. At the signal level, only the endDeviceAsset may be used as an eiTarget sub-element. The allowable values for the eiEventSignal:eiTarget:endDeviceAsset:mrid are as follows: • Thermostat • Strip_Heater • Baseboard_Heater • Water_Heater • Pool_Pump • Sauna • Hot_tub • Smart_Appliance • Irrigation_Pump • Managed_Commercial_and_Industrial_Loads • Simple_Residential_On_Off_Loads • Exterior_Lighting • Interior_Lighting • Electric_Vehicle • Generation_Systems • Load_Control_Switch • Smart_Inverter • EVSE • RESU • Energy_Management_System • Smart_Energy_Module • Storage • x-{user Defined} If more that one endDeviceAsset:mrid value is present at the signal level, the values are OR'd together to determine a match with the VEN's resources. If present, eiEventSignal:eiTarget:endDeviceAsset:mrid values are AND'd with any event level eiTarget values to determine the intended target for the specific signal. If no eiTarget:endDeviceAsset sub-element is present in the eiTarget, the eiEventSignal:eiTarget element MUST be omitted from the payload.
112	**VEN, EiEvent Service, oadrDistributeEvent payload** If a VEN receives an event where the nature of the signal type causes the VEN not to be able to resolve the intended resource to apply the event to, the VEN MUST return a 469 error response to the VTN.
113	**VTN, EiEvent Service, oadrDistributeEvent payload** The event currentValue element is optional for the B service, but MUST be included in payloads sent to A Profile VENs

114	**VEN, EiEvent Service, oadrDistribute payload** A VEN MUST have reasonably synchronized clocks with the VTN. It is up to the deployment to define the acceptable skew. It is out of scope of OpenADR how to synchronize clocks; mechanisms such as NTP may be used. When a 2.0b VEN receives an event, it MAY ignore the eventStatus contained in the event (with the exception of “cancelled”) and calculate the eventStatus based on the current time, event start time, interval duration etc.
115	**VTN, EiEvent Service, oadrDistribute payload** createdDateTime MUST be recreated each time any element of the event payload is changed. The eventStatus MUST be updated before sending out the event when a VTN receives an oadrRequestEvent. It MAY update the status when it receives an oadrPoll, but this is not required, as the 2.0b VEN can calculate the eventStatus automatically. When oadrDistributeEvent payload contains multiple events, all createdDateTime found in the payload MUST be identical.
116	**VTN/VEN, EiEvent Service, oadrDistributeEvent- Payload** The signalType element contained in a SIMPLE signal MUST be “level” for the 2.0b profile implementations. Refer to rule 69 for 2.0a profile VENs.

11.2.3 EiOpt

200	**VEN/VTN, EiOpt Service, oadrCreateOpt Payload** oadrCreateOpt payload vavailability properties MUST NOT include the following elements: • tolerance • eiRampUp • eiRecovery • eiNotification
201	**VEN/VTN, EiOpt Service, oadrCreateOpt Payload** The following aspects of an opt schedule are program specific: • The type of schedules that a VEN is allowed to issue: optIn, optOut, or both • The default opt state (optIn or optOut) for time periods not defined by either an opt or Avail schedule (if defined) • Whether optIn or optOut schedules have precedence in the situation where they have overlapping time frames • Whether overlapping time frames are allowed within the same schedule (optIn or optOut) and if they are not allowed, the expected exception handling behavior • Whether the VTN is allowed to send events during an optOut period
202	**VEN, EiOpt Service, oadrCreateOpt Payload** If the oadrCreateOpt eiTarget is empty, then the opt schedule applies to all of the resources associated with that VEN. If eiTarget sub elements are defined, these sub elements MUST be OR'd together to define a subset of the VENs resources that the opt schedule must apply. Note that the VEN MAY send multiple opt schedule payloads for different sets of resources identified by the eiTarget element.
203	**VTN, EiOpt Service, oadrCreateOpt Payload** A new opt schedule (optIn or optOut) sent via oadrCreateOpt MUST be handled by the VTN as follows when a previously sent opt schedule is still active: • If only the venID is specified in eiTarget, previous opt schedules with respect to future availability is replaced in its entirety by the new opt schedule for all resources associated with the VEN. • If sub elements are specified in eiTarget in addition to the venID, then an opt schedule MUST be generated for these resources superseding any previously defined schedules for specific resources. • Previously defined opt schedules for resources that are not specified by eiTarget MUST remain unchanged. Note that if both optIn and optOut schedules exist, they are treated independantly with respect to this rule.
204	**VEN, EiOpt Service, oadrCreateOpt Payload** The VEN MUST send an optID value as part of the oadrCreateOpt payload, however the optID value is not required to be unique, and in fact MAY be the same for all oadrCreateOpt payloads. Note that if the same optID is used defining opt schedules for multiple sets of resources, a cancelation referencing the optID MUST cancel the opt schedule for all the resources.

205	**VTN, EiOpt Service, oadrCreateOpt Payload** If marketContext is present in the oadrCreateOpt payload, the opt schedule MUST only apply to the VENs availability with respect to events generated within the specified marketContext.
206	**VTN/VEN, EiOpt Service, oadrCreateOpt Payload** A VEN responds with an EiEvent oadrCreatedEvent payload each time it receives a oadrDistributeEvent payload if oadrResponseRequired is set to “always”. However, further qualifications of the optType state for the event can be accomplished by sending either an oadrCreatedEvent or an EiOpt oadrCreateOpt payload with the eventID specified, provided that the oadrResponseRequired element was set to ‘always’ when the event was last received from the VTN. Once the optType state for a certain resource is qualified with oadrCreateOpt, subsequent changes to the optType state for the specific resource can only be made by another oadrCreateOpt payload. optType states for resources not specifically targeted in the oadrCreateOpt payload would still be controlled by subsequent oadrCreateEvent optType changes. Modifications to an event do not change this rule. Note that oadrCreateOpt may not be sent to qualify an event if oadrResponseRequired is set to “never” in the most recently received oadrDistributeEvent. If oadrCreateOpt is used to modify the opt state of a pending or active event, the following rules MUST apply with regards to the eiTarget and oadrDeviceClass targeting elements: • If no target is present in the oadrCreateOpt:eiTarget or if the only target is the venID, then all resources associated with the qualifiedEvent MUST have its opt state altered. • If the oadrCreateOpt:eiTarget further qualifies any resources of the VEN for the given qualifiedEvent (in addition to the implicitly or explicitly targeted VEN), the included resources MUST be a subset of the resources targeted in the qualifiedEvent, and the opt state will only be altered for the targeted subset. • If the VEN includes one or more targets in the oadrCreateOpt:eiTarget payload that are not part of the qualifiedEvent, the VTN MUST generate a 4xx error in its oadrCreatedOpt:eiResponse. • The VTN MAY respond with a 4xx error in the oadrCreatedOpt response if it does not accept the requested opt state changes for the specified target resources. Note that the VEN MAY send multiple oadrCreateOpt payloads with eventID specified for different sets of resources identified by the eiTarget element. When eventID is specified in oadrCreateOpt, the VEN MUST NOT include a vavailablity or marketContext in the payload. When eventID is *not* specified in oadrCreateOpt, the VEN MUST include a vavailability and MAY include a marketContext in the payload. Note that oadrCreateOpt with an eventID specified MAY also include oadrDeviceClass to further qualify the resources targeted. Resolution of the targeted resources SHOULD follow conformance rule 209.

207	**VTN/VEN, EiOpt Service, oadrCreateOpt Payload** Although requestID is a mandatory payload element for oadrCreateOpt and oadrCancelOpt, it MAY be left as an empty string. However, if a value is specified for requestID, the VTN MUST return that value in its oadrCreatedOpt or oadrCanceledOpt payload.
208	**VEN, EiOpt Service, IDs** The following ID's MUST be specified by the VEN and SHOULD be validated by the VTN receiving the payload. If the ID does not match the expected ID, the device SHOULD include a 452 error in eiResponse. • venID • optID
209	**VEN/VTN, EiOpt Service, oadrCreateOpt Payload** The oadrCreateOpt:oadrDeviceClass element, if present in a payload, MUST contain only the endDeviceAsset sub-element whose values MUST conform to the list of device classes shown in conformance rule 111. If no oadrCreateOpt:oadrDeviceClass:endDeviceAsset sub-element is present in the oadrCreateOpt, the oadrCreateOpt:oadrDeviceClass element MUST be omitted from the payload. The device classes specified in oadrDeviceClass SHOULD be AND'd with the resource targets defined by oadrCreateOpt:eiTarget and marketContext (Rule 205), to determine the resources whose availability is impacted by this opt schedule.
210	**VEN/VTN, oadrCreateOpt Payload** If an opt availability schedule uses a duration of zero it MUST be treated as an open ended opt state.
211	**VEN/VTN, oadrCancelOpt** A VEN MUST NOT use oadrCancelOpt with an optID matching that of a oadrCreateOpt used to qualify an active or pending event. An opt cancellation in this context would be illogical. VTN MUST NOT generate an error if it receives a cancellation for an opt schedule that has been superseded by a more recent opt schedule

11.2.4 EiReport

300	**VEN/VTN, EiReport Service** An alliance report profile is a subset of all the possible oadrReport object schema elements that define the characteristics (data points) of a specific type of report (history and telemetry usage, telemetry status). The Metadata report defines the characteristics that a particular implementation is capable of reporting. This MUST be a subset of the characteristics defined by a well-known alliance report profile if the reportName matches that defined by a well known report. A report request (oadrReportRequest object) defines the specific characteristics that an implementation would like to receive in a report. This list of characteristics MUST be a subset of the characteristics defined in the metadata report. A non-Metadata report (oadrReport object) MUST contain values for all the characteristics requested in the report request.
301	**VEN/VTN, EiReport Service** Implementations MUST at a minimum support the following payload interactions even if the implementation does not have any reportable data: • Send oadrRegisterReport payload on power up (after initial registration), reset, or re-registration For push implementations oadrRegisterReport would be returned by the VTN in response to an initial oadrPoll payload • Send oadrCreatedReport payload in response to an oadrCreateReport payload requesting a Metadata report • Send oadrRegisterReport payload containing a Metadata report should an oadrCreateReport payload request Metadata reports. In each case where the implementation does not have any reportable data, the oadrReport object MUST be omitted from the oadrRegisterReport payload noted above.
302	**VEN/VTN, EiReport Service** All reports are the aggregate of the resources defined in the Metadata report element reportDataSource, filtered by the optional inclusion of a marketContext in the Metadata report characteristics. If multiple target types are specified in reportDataSource, the resources targeted by these types are OR'd together.
303	**VEN/VTN, EiReport Service** Although requestID element is a mandatory payload element for request payloads, it MAY be left as an empty string. However, if a value is specified for requestID in the request payload, the responding implementation MUST return that value in its response payload.

304	**VEN/VTN, EiReport Service, IDs** The following IDs SHOULD be validated by the implementations receiving the payload. If the ID does not match the expected ID, the device SHOULD include a 452 error in the eiResponse payload. • venID • vtnID • rID • reportRequestID • reportSpecifierID
305	**VEN/VTN, EiReport Service, Metadata oadrReport Object** Payloads containing a Metadata report that describes reporting capabilities MUST include the following optional payload elements: • reportDescription • reportDescription:itemBase substitution group (Units) as indicated in conformance rule 331 For payloads containing a Metadata report that depicts that no reporting capabilities are supported, the oadrReport object MUST be omitted. Payloads containing a Metadata report MUST NOT include the following optional payload elements: • intervals • dtstart • ei
306	**VEN/VTN, EiReport Service, Metadata oadrReport Object** An implementation (VEN or VTN) sending a payload containing a Metadata report MUST supply all of its reporting capabilities. An implementation receiving a Metadata report MUST use this report to replace any previously received Metadata reporting capabilities. It MUST also implicitly cancel all previously scheduled reports, except for periodic requests for metadata reports, which MUST be explicitly canceled. If implementations receiving a Metadata report wish to continue to receive a previously requested report(s), it MUST send a new report request after the receipt of the Metadata report.
307	**VEN/VTN, EiReport Service, Metadata oadrReport Object** The Metadata oadrReportDescription:reportSubject element, if present in a payload, MUST contain only the endDeviceAsset sub-element whose values MUST conform to the list of device classes shown in conformance rule 111. If no oadrReportDescription:reportSubject:endDeviceAsset sub-element is present in the oadrReportDescription, the oadrReportDescription:reportSubject element MUST be omitted from the payload. The device classes specified in reportSubject SHOULD be AND'd with the resource targets defined by reportDataSource and marketContext (Rule 302), to determine the specific target whose data will be aggregated for the report.

308	**VEN/VTN, EiReport Service, Metadata oadrReport Object** All reportSpecifierID element values MUST be unique within a specific Metadata report. If a reportDescription in the Metadata report has a reportName element that contains one of the well known alliance report profile names, the data points characteristics and allowable values MUST conform to the table shown in conformance rule 331. All rID element values MUST be unique within a specific reportSpecifierID. When requesting a Metadata report by sending an oadrCreateReport, rID MUST be set to 0. The receiving party of the oadrCreateReport requesting a Metadata report, MUST ignore the rID value.
309	**VEN/VTN, EiReport, Metadata report – Power Up and Reset** Upon power-up or after a system reset (followed by an initial registration) PUSH and PULL implementations of both VTN and VEN MUST send an oadrRegisterReport payload (i.e., Metadata reports) to the other party prior to initiating other non-registration service operations except an oartRequestEvent payload which may be sent prior to the exchange of metadata reports.
311	**VEN/VTN, EiReport Service, Metadata oadrReport Object** The oadrRegisterReport.oadrReport.reportRequestID element in each contained metadata report MUST be set to zero. If the oadrRegisterReport payload is being sent to as the result of a oadrCreateReport request, the optional oadrRegisterReport.reportRequestID element must be present with a value that matches the reportRequestID used when the request was made. Otherwise, oadrRegisterReport.reportRequestID MUST NOT be included in the oadrRegisterReport payload.
312	**VEN/VTN, EiReport Service, Metadata oadrReport Object** The oadrReport:duration element communicates the amount of data that is available for reporting. For telemetry reporting this may reflect the amount of data that can be buffered by the VEN. For history and forecast reports the amount of available data is more likely bounded by off device storage. This element MUST be included in Metadata reports. Since duration is a time parameter the assumption is that the implementation can store enough data at the fastest supported sampling rate (oadrSamplingRate:oadrMinPeriod) for that duration of time. The oadrSamplingRate:oadrMaxPeriod MUST NOT be larger than the duration value noted above.
313	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** Payloads containing a Non-Metadata report MUST include the following optional payload elements: • dtstart • Intervals • reportName – the same value as used in the metadata report from which the report is derived, except for without the METADATA prefix. Payloads containing a Non-Metadata report MUST NOT include the following optional payload elements: • oadrReportDescription

314	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** oadrReport:dtstart element is the start time of the overall report and is inherited by the first interval in the report. If both the oadrReport:dtstart element and interval dtstart element values are present in the payload, then oadrReport:dtstart element MUST equal the dtstart element value of the first interval. Note that oadrReport:dtstart MUST be included in all oadrUpdateReport payloads. If one interval (oadrReport:intervals:interval) contains a dtstart element, all intervals MUST contain a dtstart element. Each interval MUST have an effective dtstart time, either inherited from oadrReport:dtstart or specified as part of the interval dtstart element. The sequence of interval (oadrReport:intervals:interval) available for reporting MUST be viewed as a continuum of values from the past to the future. The first interval in a sequence MUST always be the one closest to Epoch (1/1/1970), with subsequent intervals moving towards future relative to the time stamp of the first interval. For instance, an interval with a dtstart of 17:00 and a granularity of 10 minutes would cover the period from 17:00 to 17:10, followed by another interval with a dtstart 17:10 covering the period from 17:10 to 17:20.
315	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** The oadrReport:duration element is the duration of the entire report and if included MUST reflect the time period beginning of the first interval and the end of the last interval included in the report. If oadrReport:intervals:interval:duration element is absent from the payload, then the interval data contains data points that are bound to a specific point in time as opposed to a span of time. When oadrReport:intervals:interval:duration element is present, the reported dtstart time shall represent the beginning of the data collection period for a given data point. For instance, a data sample with a dtstart of 17:00 and a granularity of 10 minutes would cover the period from 17:00 to 17:10. If one interval (oadrReport:intervals:interval) contains a duration element, all intervals MUST contain a duration element. If intervals do not contain dtstart elements and there is more than one interval, duration MUST be specified in oadrReport:intervals:interval:duration.
316	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** The uid element is required in each interval if dtstart is not part of the interval specification. Within a single oadrReport object, when required, uid MUST be expressed as an interval number with a base of 0 and an increment of 1 for each subsequent interval. Exception: Green Button Reports are not required to have a uid element.

317	**VEN/VTN, EiReport Service, Non-Metadata oadrReportRequest Object** The reportSpecifier:granularity element value specified in the report request defines the requested interval reporting frequency. Example: Telemetry report with a granularity of 10 minutes with a reportBackDuration of 60 minutes would result in a report with 6 intervals. The number of intervals in a report may not exactly match the relationship between granularity and reportBackDuration, although in general it SHOULD be within plus or minus one interval of this calculated value. The requested granularity MUST NOT be less than oadrSamplingRate:oadrMinPeriod, if specified in the metadata report. At the same time, granularity MUST NOT be more than reportBackDuration. If reportSpecifier:granularity is zero then the requested data SHOULD be included in a report only when it changes from the previous value and not at regular intervals, while reports are to be sent at regular interval defined by reportBackDuration. In no case will the duration between subsequent values in a report be less than oadrSamplingRate:oadrMinPeriod, even if they are being reported only on change. In case the requested data has not changed since the last time a report has been sent out, the oadrDataQualityType MUST be set to “No New Value - Previous Value Used”. In case reportBackDuration is set to zero in the same report request, regardless of whether granularity is zero or not, the expected behavior is the one defined in Conformance Rule 324. For all history reports (HISTORY_XXX), granularity MUST be set to 0 to signify that the data SHOULD be reported at whatever granularity it was recorded.
318	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** If an implementation has no valid data for a data point which was included in its Metadata report and it has been asked to report this value via a report request, the implementation MUST include a placeholder value in the report and set the intervals:interval:oadrReportPayload:dataQuality element to one of the “Quality Bad” enumerated values.
319	**VEN/VTN, EiReport Service, Non-Metadata oadrReport Object** If included in a payload, the intervals:interval:oadrReportPayload:accuracy element MUST be in same units as the payloadFloat value for the Interval. When present with Confidence, indicates the likely variability of the prediction. This rule is not applicable if the payloadResourceStatus is used to communicate the payload value.
321	**VEN/VTN, EiReport Service, oadrReportRequest Object** Payloads containing the oadrReportRequest object MUST NOT include the following optional element: • reportSpecifier:reportInterval:properties:tolerance • reportSpecifier:reportInterval:properties:eiNotification • reportSpecifier:reportInterval:properties:eiReampUp • reportSpecifier:reportInterval:properties:eiRecovery • reportSpecifier:specifierPayload:itembase

322	**VEN/VTN, EiReport Service, oadrReportRequest object** The reportRequestID element value MUST be unique within the scope of the VEN/VTN.
324	**VEN/VTN, EiReport Service, oadrReportRequest Object** The reportSpecifier:reportBackDuration element specifies how frequently the report should be sent. If reportBackDuration is 0, then the report MUST be sent immediately and only once using the oadrUpdateReport or oadrRegisterReport payloads. History reports MUST set reportBackDuration to 0. For Telemetry reports with a reportBackDuration of 0, the granularity value MUST be ignored and only one interval of data for each data point MUST be returned in the report. If the reportSpecifier:reportBackDuration element is non-zero, the report MUST be delivered periodically using the oadrUpdateReport or oadrRegisterReport payloads. Both Metadata and non-Metadata reports MAY be delivered periodically. The number of reports delivered over the lifetime of a periodic report MAY NOT exactly match the relationship between the requested overall reporting duration and reportBackDuration. HISTORY_USAGE and periodic TELEMETRY_USAGE and TELEMETRY_STATUS reports MUST specify values for reportSpecifier:reportInterval dtstart and duration sub elements. One-off telemetry reports with point data do not require these elements and they MUST be ignored if included in the report request payload.
325	**VEN/VTN, EiReport Service, oadrReportRequest Object** For TELEMETRY_XXX reports the reportSpecifier:reportInterval:properties:dtstart element value indicates when the first in a series of one or more reports must begin. The reportSpecifier:reportInterval:properties:duration element value indicates the time span from the dtstart time that this report should cover. For all ongoing telemetry reports (TELEMERY_XXX), this duration may be in the future and once this time span has expired the report SHOULD no longer be generated. If reportSpecifier:reportInterval:properties:duration value is 0 then the report MUST be generated indefinitely. For all history reports (HISTORY_XXX), this duration is in the PAST and reflects the historical data requested. If this value is 0 then the report MUST include all the history from dtstart time.
327	**VEN/VTN, EiReport Service, oadrReportRequest Object** Payloads containing the oadrReportRequest object MUST use the well-known string “METADATA” as the reportSpecifierID element value to request a Metadata report.

328 **VEN/VTN, EiReport Service, oadrUpdatedReport Payload**
VTN or VEN MAY include oadrCancelReport object in oadrUpdatedReport payload to enable the implementation receiving the report to cancel a report as part of its response. When receiving such a payload, VTN or VEN MUST process it as defined in Section 8.3.2.3. The reportRequestIDs that appear in the oadrCancelReport object MUST be a subset of the reportRequestIDs from the oadrUpdateReport that is being responded to.

329 **VEN/VTN, EiReport, oadrCanceledReport and oadrCreatedReport**
oadrPendingReports element MUST contain a list of reportRequestID element values that includes all reports that are scheduled for future delivery.

330 **VEN/VTN, EiReport, oadrReport**
The rID element used in referencing data points is defined as a string in the OpenADR Alliance schema. The Energy Interoperation schema restricts the use of rID to three numeric digits. Implementers not concerned with using descriptive rID values MAY wish to stay within the bounds of the Energy Interoperation restrictions.

331 **VEN/VTN, EiReport, oadrReport**
The OpenADR alliance has defined a number of well-known reports. These reports are identified by the well-known names shown under the reportName column in the table below. When the well-known report names are used as part of the oadrRegisterReport, they MUST be prefixed with “METADATA_” and the values shown in the table MUST be used for the specified report.

reportName	reportType	Units	readingType	streamPayloadBase payloadBase
TELEMETRY_USAGE	usage	powerReal OR energyReal OR pulseCount	Direct Read	oadrReportPayload payloadFloat
TELEMETRY_STATUS	x-resourceStatus	None	x-notApplicable	oadrReportPayload oadrPayloadResourceStatus
HISTORY_USAGE	usage	powerReal OR energyReal OR pulseCount	Direct Read	oadrReportPayload payloadFloat

Users MAY define their own custom reports by defining a reportName that is unique to their deployment and by using any of the schema supported values to define the report.

333 **VEN, EiReport Service, Metadata oadrReport Object**
The reportDescription:oadrSamplingRate element is REQUIRED for any of the telemetry reports (TELEMETRY_XXX).

334	**VEN, EiReport Service, TELEMERTY_STATUS reports** TELEMETRY_STATUS reports do not explicitly list a Unit in their metadata specification. VENs SHOULD include each of the following optional oadrPayloadResourceStatus sub-elements in TELEMETRY_STATUS reports if appropriate for the targeted resource. • oadrLoadControlState :oadrCapacity • oadrLoadControlState :oadrLevelOffset • oadrLoadControlState :oadrPercentOffset • oadrLoadControlState :oadrSetPoint All intervals MUST have the same set of oadrLoadControlState child elements in a given report. The LoadControlState attributes are meant to mirror the LOAD_CONTROL signal attributes meaning that for each LOAD_CONTROL signal type there is a corresponding attribute in the LoadControlState of a report. Their intended use is to enable a VEN to report on its load control state that corresponds to a LOAD_CONTROL signal that might be sent to a VEN. Such reports can be used to signify the VEN's current LOAD_CONTROL state and might be used by the VTN to determine how sending a LOAD_CONTROL signal to the VEN will affect the VEN. It can also be used by the VTN as a means to verify if previously sent LOAD_CONTROL signals have been effected desired changes in the VEN. This is analogous to the correspondence between LOAD_DISPATCH signals and telemetry usage reports from VEN.
335	**VEN/VTN, EiReport Service,** oadrUpdatedReport A report cancellation MAY be included in the oadrUpdatedReport response. If the other party continues to send reports after this cancellation, the party requesting the cancellation MUST use oadrCancelReport to rerequest the cancellation.
336	**VEN/VTN, EiReport Service,** oadrCanceledReport An oadrCancelReport MAY include multiple reportRequestID values. If the receiving party cannot successfully cancel all the reports listed or does not recognize one of the included reportRequestID values, it MUST return a 4xx error in the responseCode element of oadrCanceledReport.
337	**VEN/VTN, EiReport Service** If an implementation offers a TELEMETRY_USAGE report that contains interval data points (representing a value accumulated over time), the implementation MUST sample this data periodically such that it can respond to a one-shot report request immediately with the most recent sample it has in its buffer. If the metadata report for the telemetry report offered a range of sampling frequencies, the report MUST include the interval duration in the one-shot interval report.
338	**VEN/VTN, EiReport Service, oadrReportRequest object** The reportSpecifier:specifierPayload:readingType values MUST be set to x-notApplicable. The readingType element is required by the EI schema in this location, but is not functionally used by OpenADR.

339	**VEN/VTN, EiReport Service** eiReportID is a unique identifier for a specific instance of a report. It is not used by OpenADR and SHOULD be ignored by VTN/VENs.
340	**VEN/VTN, EiReport Service** A source party SHOULD NOT request more data than is offered. Example: If a target party indicates in oadrReport:duration that it maintains 1 day of data, a source party SHOULD NOT request 2 days of data. If a source party requests more data from the target party in an oadrReportRequest (e.g., based on the requested duration), the target party MUST return an application error code 454 in an oadrCreatedReport:eiResponse.
341	**VEN/VTN, EiReport Service** oadrCreateReport:reportInterval:dtstart MUST reflect the report interval in the desired set of data that is closest to Epoch time. The dtstart time of the first interval (oadrReport:intervals:interval:dtstart) in the corresponding oadrUpdateReport payload may not exactly match the requested dtstart time (oadrCreateReport:reportInterval:dtstart), although in general the offset SHOULD NOT be greater than the value of granularity. In the case of periodic report, it is possible that a final report may be delivered slightly beyond the effective end time of the requested reporting period (i.e., the sum of overall reporting dtstart and duration).
342	**VEN/VTN, EiReport Service** The following combinations of dtstart and duration in an oadrUpdateReport intervals MUST NOT be used (otherwise an implementation would have to derive sample time for point data or the time component of interval data): 1) Interval data with just dtstart values 2) Point data with just duration values 3) Point data with dtstart and duration values If intervals contain both point data and interval data, both dtstart and duration can be specified.
343	**VEN/VTN, EiReport Service** If reportDataSource, reportSubject, and marketContext are omitted from the metadata report, the source of the report data is all resources associated with the VTN or VEN sending the metadata report.
344	**VEN/VTN EiReportService, oadrRegisteredReport Payload** VTN or VEN MAY include oadrReportRequest object in oadrRegisteredReport payload (Piggyback Report Request). VEN and VTNs are not required to implement sending Piggy Back Report Requests, but they must be able to receive such a request and act upon it.

345	**VEN/VTN EiReportService, oadrCancelReport** If reportToFollow is set to true by the report requester in oadrCancelReport object canceling periodic reports, the report sender MUST send one final additional report to the report requester.
346	**VEN/VTN, EiReportService, oadrUpdateReport** VEN SHOULD send oadrUpdateReport payloads in a predictable sequence and with consistent number of intervals per periodic report under normal operations. However, the VEN MAY send oadrUpdateReport payloads as needed (at a shorter interval than one specified by reportBackDuration) to provide corrected data (e.g., when previous value sent was of bad quality) or missing data (e.g., when measurement was delayed and thereby was not included in the previous report) to the VTN, provided that the number of intervals in the corrected data match the original report request and that all the data contained in the report replace any previously sent report values. It is not a protocol violation for the same report to be sent more than once.

11.2.5 EiRegisterParty

400	**VTN/VEN, EiRegisterParty Service, IDs** The following ID's SHOULD be validated by the implementations receiving the payload. If the ID does not match the expected ID, the device SHOULD include a 452 error in eiResponse. • venID • vtnID • registrationID
401	**VEN/VTN, EiRegisterParty, oadrCreatePartyRegistration** oadrQueryRegistration and oadrCreatePartyRegistration MUST only be used in the VEN to VTN direction. oadrRequestReregistration MUST only be used in the VTN to VEN direction. oadrCancelPartyRegistration MAY be used in either the VEN to VTN or VTN to VEN direction. A VEN whose registration has been canceled MAY attempt to initiate a new registration at a deployment specific interval. An unregistered VEN that is powered up or reset MUST attempt to initiate a registration if appropriately configured to communicate with the VTN.
402	**VEN/VTN, EiRegisterParty, oadrCreatePartyRegistration** oadrCreatePartyRegistration MUST include the following payload elements: • If the oadrTransportName is simpleHTTP, then oadrHttpPullModel MUST be set to true or false • If oadrHttpPullModel is set to false, indicating a HTTP PUSH exchange model, the oadrTransportAddress MUST be included • registrationID – when reregistering only • venID – when reregistering. If the venID is included in the initial registration payload, the assumption shall be that the VEN has been preconfigured with this value and the VTN SHOULD validate the venID just as it would any other payload with a venID oadrCreatePartyRegistration MUST NOT include the following payload elements: • registrationID – for a new registration

403	**VEN/VTN, EiRegisterParty, oadrCreatedPartyRegistration** When responding to an oadrCreatePartyRegistration.... oadrCreatedPartyRegistration MUST include the following payload elements: • All supported profiles and transports in the oadrProfiles object • If the VEN has registered with an HTTP PULL model, then the oadrRequestedOadrPollFreq MUST be included in the payload • Any oadrServiceSpecificInfo or oadrExtensions required to insure interoperability over the profile and transport being utilized In addition, oadrCreatedPartyRegistration MUST include the following payload elements, unless the oadrCreatedPartyRegistration is sent as a response to an oadrQueryRegistration (in which case inclusion of the elements is optional): • registrationID • venID
404	**VEN/VTN, EiRegisterParty, oadrCreatedPartyRegistration** When responding to an oadrQueryRegistration.... oadrCreatedPartyRegistration MUST include the following payload elements: • All supported profiles and transports in the oadrProfiles object • All relevant oadrServiceSpecificInfo or oadrExtensions that may influence the VENs choice of profile or transport. oadrCreatedPartyRegistration MUST NOT include the following payload elements: • registrationID – If the VEN has not registered with the VTN yet • venID – If the VEN has not registered with the VTN yet Note: It is not necessary for the VTN to include the oadrPoll polling information in the query response, although it may do so.
405	**VTN/VEN, EiRegisterParty, oadrCreatePartyRegistration** 2.0b VEN MUST implement EiRegisterParty service. When a 2.0b VEN boots or is reset, it SHOULD initiate registration (using the EiRegisterParty service) before sending any other message to or accepting any message from a VTN. However, the VEN MAY be registered out-of-band instead of using EiRegisterParty. A VEN SHOULD ignore all payloads from a VTN when not in a registered state. Refer to rule 517 for behavior when a VEN attempts to communicate with a VTN in an unregistered state. After a completed new registration, VENs and VTNs MUST exchange their Metadata reports, and VENs (both PULL and PUSH models) must use oadrRequestEvent to obtain all relevant events prior to initiating other payload exchanges. The exchange of MetaData reports and the use of oadrRequestEvents is not required for re-registration.

406	**VTN/VEN, EiRegisterParty, oadrCreatePartyRegistration** A VEN MAY send an oadrCreatePartyRegistration (without venID and registrationID) even if it is already registered. If a VTN receives such a new registration (that is not a requested re-registration), it MUST erase all existing reporting and registration information and initiate a completely new registration. It MAY reuse the same venID and registrationID that the VEN had before this new registration.
407	**VTN/VEN, EiRegisterParty, oadrCancelPartyRegistration** If a device receives an oadrCancelPartyRegistration from the other party, it MUST erase all information about this device, including exchanged Metadata reports, requested reports, and registration information.

11.2.6 General Conformance Rules

500	**VEN/VTN, oadrPoll** oadrPoll is a service independent polling mechanism used by VENs in a PULL model to request pending service operations from the VTN. oadrPoll MUST NOT be used by VENs in a PUSH model. oadrPoll MUST NOT be used in the VTN to VEN direction. When using oadrPoll, the rules for which payloads are valid and how those payloads delivered are the same as if the VTN had initiated the operations and pushed the payloads to the VEN. Only one operational payload MAY be sent by the VTN in response to the oadrPoll message. When no additional operational payloads are available in the queue, the VTN will respond with an oadrResponse payload. At each polling, VENs SHOULD, without waiting until the next polling interval, continue sending oadrPoll messages and processing payloads until an oadrResponse payload is returned to signal an empty VTN message queue. If the oadrPoll contains an incorrect venID, the VTN MUST respond with an oadrResponse containing a 452 error code. If a logical response is required by the VEN to the received operational payload, the VEN MUST send that logical response asynchronously via a transport request. The VTN should acknowledge this logical response with an oadrResponse payload. The VTN MAY optionally ignore an oadrPoll if it has not received an expected logical response to a payload delivered as a response to a previous poll. The VTN should be coded such that after some timeout it gives up waiting for the expected response and resumes responding to oadrPoll oadrPoll requests MUST be used by a VEN to retrieve messages from the VTN. The polling interval (i.e., the amount of time between two successive polls, specified as ISO 8601 duration in oadrRequestedOadrPollFreq) SHOULD be as requested by the VTN during registration, but MAY be higher because of processing constraints of the VEN or other factors. The polling interval SHOULD NOT be lower than the requested duration in oadrRequestedOadrPollFreq, except when the VEN is emptying the VTN payload queue. Service specific requests SHOULD only be used for re-synchronization where necessary. While it is not a protocol violation for a VEN to poll at an interval longer that oadrRequestedOadrPollFreq specified by the VTN, the actual polling interval used by the VEN SHOULD NOT be greater than 20% of the average event duration. This will allow the VTN to cancel an event in a timely manner.
501	**VEN/VTN, oadrPoll** All OpenADR B profile VTN implementations as well as simple HTTP VENs MUST support oadrPoll.

502	**VEN/VTN, oadrPoll, EiEvent Service** A VTN that is polled with oadrPoll typically will return a oadrDistributeEvent only when a new event is generated or event content has changed since the last oadrDistributeEvent was transmitted to the VEN. Transition of eventStatus, except for cancellation, SHOULD NOT be considered as change of event content. However, it is not an error for an VTN to return an oadrDistributeEvent in response to oadrPoll with only events that it has previously communicated. It should also be noted that the oadrDistributeEvent payload MUST contain all active and pending events, regardless of whether it is polled using oadrPoll or oadrRequestEvent. The behavior described above is consistent with existing conformance rules and is provided here as clarification of expected behavior.
506	**VEN/VTN** A VTN that supports the B profile MUST also concurrently support the A profile. A VEN that supports the B profile MAY also support the A profile. In that case, the VEN can be configured to communicate with a 2.0a VTN using, e.g., any of the following options: - manual configuration (e.g., as part of setting up the URL of the VTN) - automatic fallback (during EiRegisterParty) when receiving any reply from the VTN using the 2.0a namespace - automatic configuration based on the URL of the VTN (which contains "2.0b" for B VTNs as per conformance rule 511)
507	**VEN/VTN, Transport** B profile VTNs MUST support XMPP in addition to the Simple HTTP transport. B profile VENs MAY support either HTTP or XMPP or both. B profile VEN and VTNs that implement XMPP MUST support the PUSH model. XMPP VENs MAY still make requests of the VTN as in the PULL model, however they MUST NOT use the oadrPoll request.
508	**VEN, venID** All payloads sent by a VEN to a VTN MUST contain a valid venID. This venID typically appears just below the root payload element, but in a few payloads (e.g., oadrRequestEvent and oadrCreatedEvent), the venID is one layer deeper in the schema. If a payload containing one of these venID elements off the root is instead sent by the VTN, the venID (if optional) MAY be omitted from the payload contents.
509	**VEN/VTN, Schema Version** The optional schemaVersion attribute MUST be included in all payloads exchanged. The value of the schemaVersion attribute MUST be either “2.0a” or “2.0b” to indicate the version of the OpenADR profile being used.

510	**VEN/VTN – Minimum B Profile Feature Support** In order to facilitate interoperability and validation of behavior, the following features MUST be supported: 1) A VTN MUST be capable of sending and a VEN MUST be capable of receiving the following standard event signals: • SIMPLE • ELECTRICTY_PRICE with a signalType of price • LOAD_DISPATCH with a signalType of setpoint 2) A VEN MUST be capable of producing the following standard report, in addition to the metadata report, which both VENs and VTNs MUST support: • TELEMETRY_STATUS report with the mandatory data points of oadrOnline and oadrManualOverride for an attached resource identified with a resourceID target A VEN MUST be capable of producing TELEMETRY_USAGE reports, at least for certification (and MAY offer it in deployments). The device MUST be able to send some telemetry data (i.e., in case it does not have any metering resources attached, it MUST provide sample data). A VEN MAY in addition support HISTORY_USAGE or other reports. A VEN MUST provide sufficient storage to store recent data, at least 100 data points. In case the last reading has not been received by the VTN (e.g., because of transitory communication problems), recent history can be requested by the VTN from the VEN. A VTN MUST support report registration (i.e., exchange of Metadata report), and MAY optionally support additional report types. VTNs are not required to support periodic reporting of metadata reports if they do not have any reports to offer. Should a VTN receive a request for a periodic metadata report and have no reports to offer, it SHOULD respond to the request as if it were a one-shot request with the reportBackDuration value set to 0. 3) A Report Only VEN MUST be capable of supporting all the Alliance defined standard reports including: • TELEMETRY_USAGE • TELEMETRY_STATUS • HISTORY_USAGE 4) A VTN MUST be capable of utilizing the eiTarget sub-elements of venID and resourceID. 5) A VEN MUST be capable of utilizing the EiOpt service to further qualify the opt state of an event. While there is the expectation that fully functional VENs will be able to generate opt schedules, and fully functional VTNs will be capable of generating baselines and more than just metadata reports, these capabilities are not explicitly defined in this rule. If these capabilities can be configured by implementations, they will be tested as part of certification.

511	**VTN/VEN** B Profile endpoints MUST use the following template in the simple HTTP mode, while accepting normalized URIs according to the rules specified in [RFC3986] (page 40). Port and prefix are optional. • https://<hostname>(:port)/(prefix/)OpenADR2/Simple/2.0b/<service> With the following strings used for the service name: • EiEvent • EiOpt • EiReport • EiRegisterParty • OadrPoll (only on the VTN) When VTN and VEN are running on the same IP addess, they MUST be deployed under different prefix while using the path under it as described above.
512	**VTN/VEN** venID and vtnID MUST be case-sensitive (e.g., a vtnID of “vtnID1” is different from “vtnid1” when parsing an incoming message). For marketContext, the normalization rules specified in [RFC3986] (page 40) MUST be applied, allowing for hostname and scheme (at least) to be case insensitive and still be equivalent.

514	**VEN/VTN, XML signatures** Implementations MAY optionally support XML signatures as defined in Section 10.6 of this specification. If a device is configured to use XML Signatures, it MUST ignore incoming messages that do not contain a valid signature, as defined in Section10.6. If supported, implementation MUST select from the following list of methods when creating the XML signature: canonicalizationMethod: http://www.w3.org/TR/2001/REC-xml-c14n-20010315 http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments SignatureMethod: http://www.w3.org/2000/09/xmldsig#rsa-sha1 http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256 DigestMethod: http://www.w3.org/2000/09/xmldsig#sha1 http://www.w3.org/2001/04/xmlenc#sha256 The SignatureMethod and DigestMethod selected SHOULD be consistent with the negotiated TLS cipher and SHOULD be base64 encoded. Note that a VTN or VEN MAY be configured to support any methods based on the needs of a specific deployment. However in the absence of changes to the default configuration of the VTN or VEN, the behavior of the devices MUST be as noted above. A device using XML signatures MUST include a ReplayProtect element as SignatureProperty (refer to Section 10.6 for an example). The ReplayProtect element MUST contain the dateTime when the payload is sent to the other party (not when it is created), as well as a random nonce. A device receiving a payload in high security mode MUST verify if the ReplayProtect element is part of the SignatureProperties element and MUST reject the payload if the current date and time on the device differs from the value in the ReplayProtect more than a predefined value. In addition, the nonce MAY be used for further protection against replay attacks.

515	**VEN/VTN, XMPP** VEN to VEN communication MUST not be allowed by the XMPP server. An XMPP client (both VTN and VEN) MUST support XMPP Presence (refer to section 9.3.4.5). The client MUST implement XMPP Ping and MAY use it in deployments (refer to section 9.3.4.6). The VEN MUST only use a single endpoint defined by a single JID. During authentication to the XMPP server, the Common Name (CN) of the x.509 certificate MUST match the username of the client. VENs MUST use the XMPP discovery mechanism to determine the service endpoints provided by the VTN
516	**VEN/VTN, Expired Certificate** VENs and VTNs shall not communicate at the application layer with a counter party that has an expired certificate.
517	**VTN, Error Codes** For communication with VEN, a VTN MUST return errors in the ways defined below. If TLS handshake fails (e.g., because of invalid or expired digital certificate), the VTN MUST return a transport layer error code of 403 for HTTP or a <forbidden/> error for XMPP. If TLS handshake is successful but the VEN is not yet registered, return a 463 application error code in an appropriate (relative to the request) OpenADR payload. If TLS handshake is successful and the VEN is registered but a wrong ID is specified in the payload from the VEN (e.g., venID different from the one set during registration), return 452 error code in an appropriate (relative to the request) OpenADR payload.

11.3 Cardinality

The OpenADR 2.0 profile utilizes a subset of the OASIS Energy Interoperation 1.0 schema. However, all root payload elements are in an OpenADR specific namespace, as are some elements required to add functionality to OpenADR 2.0 that was not supported by Energy Interop. In some cases, objects from Energy Interoperation EiEvent service could be utilized with changes in cardinality as is outlined in the Table 6.

Table 6 Cardinalities

Object	Element	EI	OADR 2.0
EiEventType	eventDescriptor:eventStatus	0 – 1	1
EiEventType	eventDescriptor:createdDateTime	0 – 1	1
EiEventType	eiActivePeriod:properties	0 – 1	1
EiEventType	eiActivePeriod:properties/dtstart	0 – many	1
EiEventType	eiActivePeriod: properties:duration	0 – many	1
EiEventType	eiActivePeriod: properties:tolerance	0 – many	0 – 1
EiEventType	eiActivePeriod:properties:x-eiNotification	0 – many	1
EiEventType	eiActivePeriod: properties:x-eiRampUp	0 – many	0 – 1
EiEventType	eiActivePeriod: properties:x-Recovery	0 – many	0 – 1
EiEventType	eiEventSignals:eiEventSignal	0 – many	1 (A profile) 1 – many (B profile)
EiEventType	eiEventSignals:eiEventSignal:intervals	0 – 1	1
EiEventType	eiEventSignals:eiEventSignal:currentValue	0 – 1	1 (A profile) 0 – 1 (B profile)
EiEventType	eiEventSignals:eiEventSignal:intervals:interval	0 – many	1 – many
EiEventType	eiEventSignals:eiEventSignal:intervals: Interval:duration	0 – many	1
EiEventType	eiEventSignals:eiEventSignal:intervals: Interval/uid	0 – many	1
EiEventType	eiTarget	0 – 1	1

11.4 Services used from OASIS Energy Interoperation V1.0 Standard

The OpenADR 2.0 A and B Profile Specifications use a number of services defined and required by the OASIS Energy Interoperation standard. The following services are currently supported:

- EiEvent
- EiOpt (not in OpenADR 2.0a)
- EiReport (not in OpenADR 2.0a)
- EiRegisterParty (not in OpenADR 2.0a)

11.5 Services not currently used from OASIS EI

The OpenADR 2.0 A and B Profile Specification do not use the following services included in the OASIS Energy Interoperation OpenADR profile:

- EiQuote
- EiEnroll
- EiAvail
- EiMarketContext

Annex A – Detailed Report Description

Report Profiles			29-Nov-12									
Attributes of oadrReport							DATA REPORTS					
							HISTORY		TELEMETRY			
							USAGE LOGS		CURRENT USAGE		RESOURCE STATUS	
						GENERAL DESCRIPTION	METADATA	REPORT	METADATA	REPORT	METADATA	REPORT
xcal:dtstart						NA	NA	*start date/time of data*	NA	*start date/time of data*	NA	*start date/time of data*
xcal:duration						This is the amount of data that may be reported expressed as a duration of time. For entities that sample data at some maximum frequency this is the amount of data that can be stored at that frequency.	*amount of history*	*duration of entire report (optional)*	*amount of data*	*duration of entire report (optional)*	*amount of data*	*duration of entire report (optional)*
strm:intervals	ei:interval There is one of these elements for each datapoint value in the report.	xcal:dtstart				Start date/time of the interval	NA	*date/time*	NA	*date/time*	NA	*date/time*
		xcal:duration				duration of the interval. If ommitted then the datapoint in this interval is for a specific period of time and does not span an interval of time.	NA	*interval duration*	NA	*interval duration*	NA	NA
		xcal:uid				sequence number of this interval starting at 0	NA	*Y*	NA	*Y*	NA	*Y*
		reportPayload	ei:rid			Identifier of the data point.	NA	*ID*	NA	*ID*	NA	*ID*
			ei:confidence			How much confidence there is in the value (0 -1)	NA	*(optional)*	NA	*(optional)*	NA	*(optional)*
			ei:accuracy			Accuracy of the data	NA	*(optional)*	NA	*(optional)*	NA	*(optional)*
			ei:dataQuality			Text describing data quality	NA	*(optional)*	NA	*(optional)*	NA	*(optional)*
			payloadFloat			Actual value of the datapoint	NA	*Usage Value*	NA	*Usage Value*	NA	NA
			oadrPayloadResourceStatus	oadrOnline		if TRUE the resource is on line.	NA	NA	NA	NA	NA	*TRUE if online*
				oadrManualOverride		if TRUE the control or state of the resource was manually overridden.	NA	NA	NA	NA	NA	TRUE if override
				oadrCapacity	oadrMin	These correspond to the LOAD_CONTROL signals. Each one has the following attributes: - oadrMin - the minimum possible value - oadrMax - the maximum possible value - oadrCurrent - the current value - oadrNormal - the normal operating value	NA	NA	NA	NA	NA	*optional*
					oadrMax		NA	NA	NA	NA	NA	*optional*
					oadrCurrent		NA	NA	NA	NA	NA	*optional*
					oadrNormal		NA	NA	NA	NA	NA	*optional*
				oadrLevelOffset	oadrMin		NA	NA	NA	NA	NA	*optional*
					oadrMax		NA	NA	NA	NA	NA	*optional*
					oadrCurrent		NA	NA	NA	NA	NA	*optional*
					oadrNormal		NA	NA	NA	NA	NA	*optional*
				oadrPercentOffset	oadrMin		NA	NA	NA	NA	NA	*optional*
					oadrMax		NA	NA	NA	NA	NA	*optional*
					oadrCurrent		NA	NA	NA	NA	NA	*optional*
					oadrNormal		NA	NA	NA	NA	NA	*optional*
				oadrSetPoint	oadrMin		NA	NA	NA	NA	NA	*optional*
					oadrMax		NA	NA	NA	NA	NA	*optional*
					oadrCurrent		NA	NA	NA	NA	NA	*optional*
					oadrNormal		NA	NA	NA	NA	NA	*optional*
ei:eiReportID						Identifier for this particular instance of the report	*NA*	*my report ID*	*NA*	*my report ID*	*NA*	*my report ID*
oadr:oadrReportDescription (There is one of these elements for each data item, e.g. data point, in the report)	ei:rID					This is the identifier for each datapoint in the report. Each datapoint has its own set of oadrReportDescription attributes which are used to describe the datapoint . When the report is requested the requesting party may specify which set of datapoints it wants in the requested report. RULE: rID's must be unique within a specific reportSpecifierID.	*ID*	ID	*ID*	ID	*ID*	ID
	ei:reportSubject (endDeviceAsset ONLY)	ei:eiTargetType				This is used to specify device classes using the endDeviceAsset attribute (optional)	Device class if applicable	NA	Device class if applicable	NA	*Device class if applicable*	NA
	ei:reportDataSource (ONE OF)	ei:eiTargetType				This is the actual source of the data and allows for the normal target types. (optional)	*resourceID*	NA	*resourceID*	NA	*resourceID*	NA
	ei:reportType					This is the type of information in the report. See ReportEumeratedType for a list of legal values.	**usage**	NA	**usage**	NA	**x-resourceStatus**	NA
	emix:itemBase (ONE OF)	current				This is a partial list of the units for the data in the report .	NA	NA	NA	NA	NA	NA
		energyApparent					NA	NA	NA	NA	NA	NA
		energyReactive					NA	NA	NA	NA	NA	NA
		energyReal					Y	NA	Y	NA	NA	NA
		powerApparent					NA	NA	NA	NA	NA	NA
		powerReactive					NA	NA	NA	NA	NA	NA
		powerReal					Y	NA	Y	NA	NA	NA
		voltage					NA	NA	NA	NA	NA	NA
		PulseCount					Y	NA	Y	NA	NA	NA
		currency					NA	NA	NA	NA	NA	NA
	ei:readingType					This is the way in which the values in the report are determined. See eiReadingTypeType for an enumeration of values.	**Direct Read**	NA	**Direct Read**	NA	**x-notApplicable**	NA
	emix:marketContext					This is the program that this report applies to. (optional)	*DR program (optional)*	NA	*DR program (optional)*	NA	*DR program (optional)*	NA
	oadrSamplingRate	oadrMinPeriod				Minimum sampling period	*min period*	NA	*min period*	NA	*min period*	NA
		oadrMaxPeriod				Maximum sampling period	*max period*	NA	*max period*	NA	*max period*	NA
		oadrOnChange				Flag to sample on value change	set to TRUE if can sample on change	NA	set to TRUE if can sample on change	NA	*set to TRUE if can sample on change*	NA
ei:reportRequestID						This is normally the ID used when this report was requested. This field should be set to 0 in the case where a METADATA report is not requested.	*0 or request ID*	*request ID*	*0 or request ID*	*request ID*	*0 or request ID*	*request ID*
ei:resportSpecifierID						This is an indentifier generated by the entity that created this METADATA report and used to refer to this specification in future report requests	*specifier ID (used by future report requests)*	specifier ID	*specifier ID (used by future report requests)*	specifier ID	*specifier ID (used by future report requests)*	specifier ID
ei:reportName						This is the name of the OADR report profile represented in this artifact.	**METADATA_HISTORY_USAGE**	**HISTORY_USAGE**	**METADATA_TELEMETRY_USAGE**	**TELEMETRY_USAGE**	**METADATA_TELEMETRY_STATUS**	**TELEMETRY_STATUS**
ei:createdDateTime						Date-time this artifact is created.	*current Date/Time*	*current Date/Time*	*current Date/Time*	*current Date/Time*	*current Date/Time*	*current Date/Time*

Annex B B Profile Extensions

B.1 Overview

The B profile schema provides a number of ways in which various enumerated values and standardized sets of values (signal and reports) can be extended without the need to modify the schema. However, utilizing the extension mechanisms may cause interoperability issues between VEN and VTNs unless both are aware of the extensions. In general, the extensions rely on a schema type called EiExtensionTokenType, which allows enumerated values to be extended with a “x-” prefix without causing schema validation errors.

B.2 Report Extension

The OpenADR Alliance has defined a number of standardized reports that are intended to meet most needs. If necessary, custom reports can be added by creating a new reportName prefixed with “x-”. Existing reportType, units (itemBase), and readingType values can be utilized to construct the new report. Custom reportType and readingType's may also be defined if necessary by prefixing the new values with “x-”.

B.3 Event Extension

The OpenADR Alliance has defined a number of standardized signals that are intended to meet most needs. If necessary, custom signals can be added by creating a new signalName prefixed with “x-”. Existing signalType and units (itemBase) values can be utilized to construct the new signal. Custom signalType can be defined if necessary by prefixing the new value with “x-”.

B.4 Other Extensions

Other enumerated schema elements that can be extended via the “x-” prefix mechanism include optReason, responseCode, oadrDataQuality, schemaVersion, and device classes defined in the endDeviceAsset element contained in the EiEvent, EiOpt, and EiReport schemas.

The VTN may also support additional extension mechanisms that can be communicated to the VEN using the oadrCreatedRegistration:oadrExtensions object. How these extensions are implemented and what functionality they may provide are outside the scope of this specification. However, great caution should be exercised in the implementation of any extension to insure interoperability with the large ecosystem of VEN implementations.

Annex C – oadrPoll Scenarios

C.1 Overview

The current conformance rules allow different scenarios when a VTN has multiple payloads in its queue and it receives oadrPoll from the VEN. This annex exemplifies several scenarios that are in accordance with this specification and thus have to be supported by implementations.

C.2 Scenarios

Scenario 1:

1. VEN sends oadrPoll
2. VTN responds with application layer request (oadrCreateReport)
3. VEN sends application layer response (oadrCreatedReport)
4. VTN responds with acknowledgement (oadrResponse)
5. VEN sends another oadrPoll
6. VTN responds with next item in the queue

It is assumed that this would be the typical behavior.

Scenario 2:

1. VEN sends oadrPoll
2. VTN responds with application layer request (oadrCreateReport)
3. VEN sends another oadrPoll
4. VTN ignores the oadrPoll and does not respond
5. VEN sends application layer response (oadrCreatedReport)
6. VTN responds with acknowledgement (oadrResponse)
7. VEN sends another oadrPoll
8. VTN responds with next item in the queue

The conformance rules allow the VTN to optionally ignore oadrPoll if a pending application layer response is expected

Scenario 3:

1. VEN sends oadrPoll
2. VTN responds with application layer request (oadrCreateReport)

3. VEN sends another oadrPoll
4. VTN responds with application layer request (oadrDistributeEvent)
5. VEN sends application layer response (oadrCreatedReport)
6. VTN responds with acknowledgement (oadrResponse)
7. VEN sends application layer response (oadrCreatedEvent)
8. VTN responds with acknowledgement (oadrResponse)
9. VEN sends another oadrPoll
10. VTN responds with next item in the queue

VEN implementations will likely behave like scenario 1 (although the other scenarios are also valid): the VEN will not poll again until it provides an application layer response to the most recent payload from the VTN.

Annex D Definition of VEN, Resource, and Party

By definition of DR, grid side entities (e.g. service providers such as utilities, ISO's, etc.) intentionally try to modify the load profiles of demand side entities (e.g. customers) through a logical construct known as a *Resource*.

A Resource is a demand side commodity that is associated with a load profile. Examples of Resources include the following:

- A single load such as a water heater.
- A collection of loads within a building such that the entire building is the Resource.
- A collection of buildings across a campus.
- A collection a disparate customers as in the case of aggregated Resources.

The key is that a Resource is associated with a load profile that may be influenced by grid side service providers. Resources may be composed of other Resources in a hierarchical fashion. An *Asset* is a special type of a Resource that does not contain any sub-Resources and typically refers to specific physical loads.

A service provider influences the load profile of a Resource by sending out "DR signals" that are "applied" to Resources. Note that this does not imply that a service provider communicates with resources.

A *Party* is an entity that enters into some sort of business relationship or contract. Service providers and customers are Parties. Resources are commodities that are "owned" by Parties.

A *VEN* is a demand side entity that is associated with Parties and Resources and forms the communications end point in the form of services that can be used to facilitate communications concerning the demand side Resources and Parties that are associated with that VEN. Note that when Resources and Parties are associated with a VEN it may take on "meaning" at a system level within the context of those associations, but the primary function of the VEN is a means of communicating about the Parties and Resources associated with the VEN.

A *VTN* is a grid side entity that is associated with service providers and forms the communications end point in the form of services that can be used to facilitate communications with demand side entities concerning Resources and Parties.