# OpenADR 3.0
# OpenADR 3.0 User Guide (non-normative)

Updated 09/06/2023
Revision Number: 3.0.0
Document Status: **Final Specification**
Document Number: 20230706-2


| Contact: | Editors: | Technical Director OpenADR Alliance: |
|---|---|---|
| OpenADR Alliance<br>111 Deerwood Road, Suite 200<br>San Ramon, CA 94583<br>USA<br>info@openadr.org | Frank Sandoval - Pajarito Technologies LLC<br>Bruce Nordman – LBNL<br>Other OpenADR Alliance Members | Rolf Bienert<br><rolf@openadr.org> |


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

# CONTENTS

# 1 Normative References

[OADR-3.0-Specification] OpenADR 3.0 OpenAPI YAML (SwaggerDoc) Specification, Draft April 17, 2023: https://github.com/oadr3/openapi-3.0.0

# 2 Informative References

[OADR-3.0-Reference_Implementation] OpenADR 3.0 Reference Implementation https://github.com/oadr3/RI-3.0.0

[OADR-3.0-Enumerations] OpenADR 3.0 Enumerations, Draft April 17, 2023

[OADR-3.0-Definition] OpenADR 3.0 Definition, Draft April 17, 2023

[OADR-2.0b-Program_Guide] OpenADR 2.0 Demand Response Program Implementation Guide, Revision Number: 1.1.2, 2017 (1.0 from 2016)

[REST_Best_Practice] RESTful web API design (website) https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

[swaggerhub] API tools https://swagger.io

# 3 Introduction

This document describes a number of common user scenarios of OpenADR 3.0, providing examples of program, event, reports, and endpoint usage. These examples are not prescriptive or normative but are provided as illustrations of how one might use the API.

This document has a similar purpose as the OpenADR 2.0b Program Guide [OADR-2.0b-Program_Guide].

# 4 Design Principles

OpenADR 3.0 was designed with the following goals and principles in mind.

## 4.1 Business objectives

- **Lower barriers to entry for new entrants.**
  REST is a more common approach for web services today than SOAP-like interfaces as embodied in OpenADR 2.0b. Therefore more programmers are familiar with more tools to work with REST APIs. It is anticipated that OpenADR will be adopted more quickly by more entities with a simple REST definition as an alternative.
- **Simpler Implementation.**
  REST interfaces are less verbose than SOAP equivalents and therefore reduce network utilization, memory footprint, and message latency. REST/JSON is easier to develop and debug than SOAP equivalents. This is particularly important for flexible loads that have a modest overall software footprint.
- **Coexistence with OpenADR 2.0b.**
  A REST format of OpenADR is significantly different from 2.0b and the formats are incompatible. It is envisioned that new entrants will develop to the REST format and therefore a growing population of 3.0 VTNs and VENs will emerge. Existing 2.0b VENs may add support for 3.0 to interoperate with newer 3.0 VTNs, and existing 2.0b VTNs may add support for REST and therefore support both 2.0b and 3.0 VENs. It is conceivable that real-time translators between the formats will be developed.

## 4.2 Design Goals

- **Functional equivalence with 2.0b.**

OpenADR 3.0 supports most, if not all, use cases supported by 2.0b and in use today. OpenADR 3.0 drops support for some use cases that are not currently or likely to be used.

- **Forward Compatibility.**
  The API will be able to add new functionality without breaking compatibility with existing clients, i.e. VTNs can evolve independently of VENs.
- **Push.**
  A pure REST implementation does not support a PUSH model, as it relies only on HTTP requests from a client, which requires clients to poll the API. In order to deliver events to clients in a low latency application, OpenADR 3.0 defines a PUSH model based on webhooks.
- **Best Practices.**
  Numerous sources publish REST best practices, and while there is no authoritative guide, [REST_Best_Practice] is a good starting point.

# 5 User Stories

A User Story is a single sentence in the form of “As an [actor,] I [want to], [so that]”. This makes explicit which entity is doing what action and the value they receive. Typically a User Story represents an action of a human, but many of these interactions are conducted automatically by software agents.

## 5.1 Actors

- Business Logic (BL): a software client of a VTN that performs operations on behalf of an energy or energy service provider.
- BL entity: an electricity provider or other business entity that provides the means to onboard VENs and energy resources represented by those VENs.
- Customer: a person that performs setup tasks prior to a VEN conducting interactions with a VTN.
- Customer Logic (CL): a software entity that embodies the functional logic within a VEN. A VEN is the software that communicates via the OpenADR 3.0 protocol, and the CL is the software, embodied within the VEN or separated via a VEN specific interface, that in turn interfaces with a set of energy resources.
- Resource. A device or system that performs operations as informed by program and events obtained by a VEN and provides data for VEN reports.

## 5.2 BL User Stories

A. As BL, I want to create, read, update, and delete program objects, so that I can implement my Demand Response (DR) programs.
B. As BL, I want to create, read, update, and delete events, so that I can influence customer energy resource behavior.
C. As BL, I want to request reports, and read and delete reports, so that I can measure and manage a DR program.
D. As BL, I want to register a callback and receive a notification when a report is created, so that I can measure and manage a DR program.

## 5.3 Customer User Stories

A. As a Customer, I want to receive a ‘Program Description’ from a BL entity, so that I can configure my VEN to interoperate with a set of programs and events available from a BL entity.
B. As a Customer, I want to receive the baseURL of the appropriate VTN from a BL entity, so that I can access the VTN to obtain programs and events and perform operations on energy resources.
C. As a Customer, I want to receive a client ID from a BL entity, so that I can use the VTN to obtain programs and events and perform operations on energy resources.
D. As a Customer, I want to receive API credentials from a BL entity, so that I can use the VTN to obtain programs and events and perform operations on energy resources.
E. As a Customer, I want to receive energy resources IDs from a BL entity, so that I can report energy resource data to a VTN..

### 5.4 CL User Stories

A. As CL, I want to read a list of programs or an individual program, so that I have the context necessary to understand what DR programs exist, or select the one appropriate to me.
B. As CL, I want to register a callback and receive a notification when a program is created, deleted, or updated, so that I have the context necessary to respond to a DR program.
C. As CL, I want to read a list of events or an individual event, so that I may respond to DR events within a program.
D. As CL, I want to register a callback and receive a notification when an event is created, updated or deleted, so that I may respond to DR events within a program.
E. As CL, I want to create and update reports, so that I can inform BL of energy resource status and behavior in response to events.

## 6 Scenarios

The following are a set of scenarios that illustrate common interactions. In the sequence diagrams, a dashed line indicates that time has passed. Solid lines indicate actions that occur close in time.

### 6.1 High Level Workflow

A VEN client generally requires security credentials to access the VTN API, therefore it is presumed that a Business Logic entity provides an onboarding process for VENs to access the API and participate in a DR program. Once the onboarding process is complete, the typical interaction pattern is for BL to create program and event resources in the VTN, and for VENs to subsequently read these resources and then generate reports. Finally, BL reads reports from the VTN. Figure 1 illustrates a typical expected sequence of interactions.

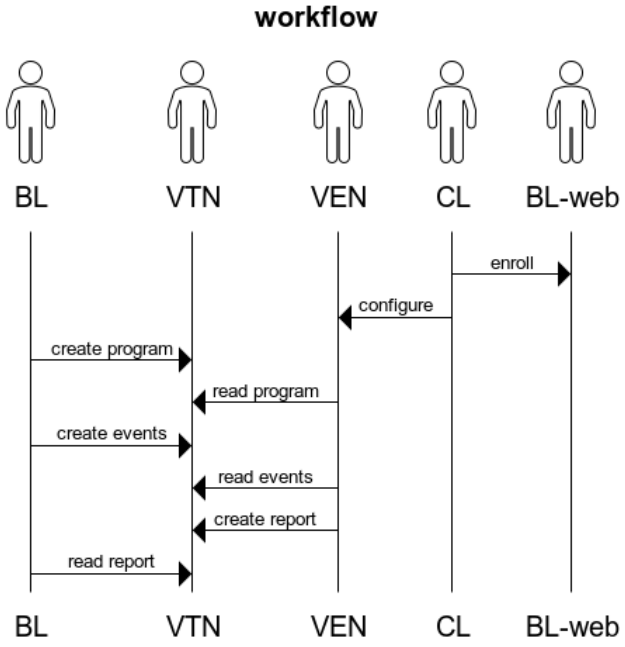



**Figure 1. High Level Workflow**

An exception to this process is for programs that are available to anyone, without a security credential. The most common case for this is ordinary tariffs, in which there is no reason to keep private the prices they include. In this case, the workflow is mostly the same except that there is no initial ‘enroll’ activity, and generally no use of reports (since the VEN is not known to the VTN). The CL does need to configure the VEN with the URL for the VTN, possibly the identity of the retailer (this may be the same for the entire VTN), and the tariff the customer is on (locational tariffs can be implemented with a simple suffix on the program ID).

### 6.2 Enrollment Scenarios

OpenADR 3.0 assumes an enrollment process has happened prior to interactions of a VEN with a VTN, in order to provision security credentials and otherwise onboard a VEN into a energy provider's set of programs. The OpenADR 3.0 standard does not define how this process is implemented. It is addressed here as background information as every energy provider must develop their own process and mechanisms to support enrollment. Figure 2 illustrates what might be expected from a typical out-of-band, unspecified enrollment flow. Note that 'register resources' and 'provide resource IDs' are optional.

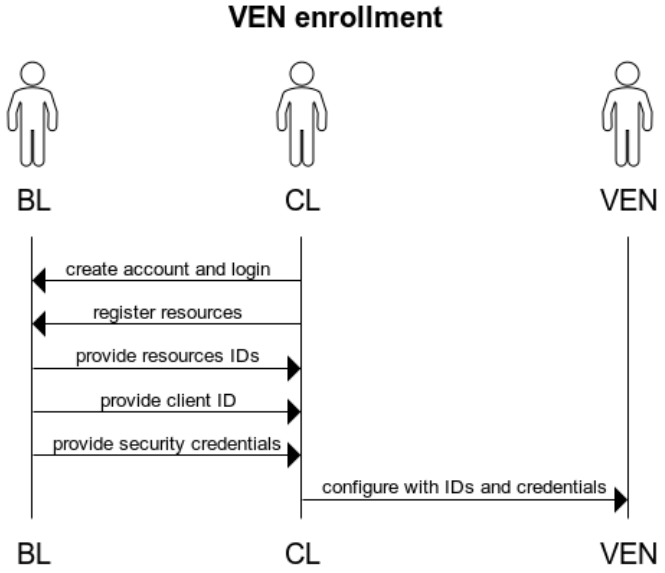



**Figure 2. Enrollment Scenario**

### 6.3 Subscription scenarios

In order to support a PUSH model, the API provides a mechanism for a client to register interest in ("subscribe to") operations on resources and receive a 'callback'. This pattern is known as a 'webhook'.

A client creates a subscription indicating a callback URL and a list of the operations and resources it is interested in. For example, a VEN client may wish to receive a callback whenever an event is created, or whenever it is created, modified, or deleted. A BL may set a subscription for reports created by VENs to be notified when a new report is created.

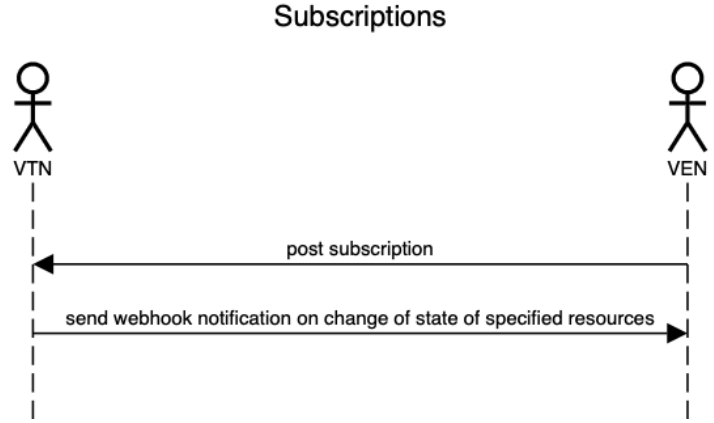



**Figure 3. Subscription scenario**

### 6.4 Program Scenarios

A program is a Demand Response offering of an energy provider. In OpenADR 3.0, a tariff is simply a type of program. Metadata about a program is represented by a program object in OpenADR 3.0 (there is no similar construct in 2.0b). Program metadata usually changes infrequently, perhaps once a year or less. Some fields in the program object may be displayed to persons using a VEN via a VEN provided user interface, but this feature is not required.

A program may declare a set of events and report types needed to meet the business objectives of the program. Every event and report is associated with exactly one program. A provider might offer several programs at the same time, such as a dynamic pricing program that executes concurrently with a load shed program. A single customer may be enrolled in multiple programs simultaneously, e.g. a battery program and an EV program.

Prior to creating events, the BL will create a program in the VTN, which VENs can query for. A VEN can also create a subscription for events, and then receive notifications of new events with a webhook. Webhooks support a ‘push’ model whereby a VEN may register a callback URL with the VTN to receive notifications. In practice, VENs may find that simple polling, or a ‘pull’ model is sufficient for receiving prices that are updated on a regular schedule, but utilize subscriptions for alerts or other events with no particular schedule. This observation applies to the other scenarios described below.

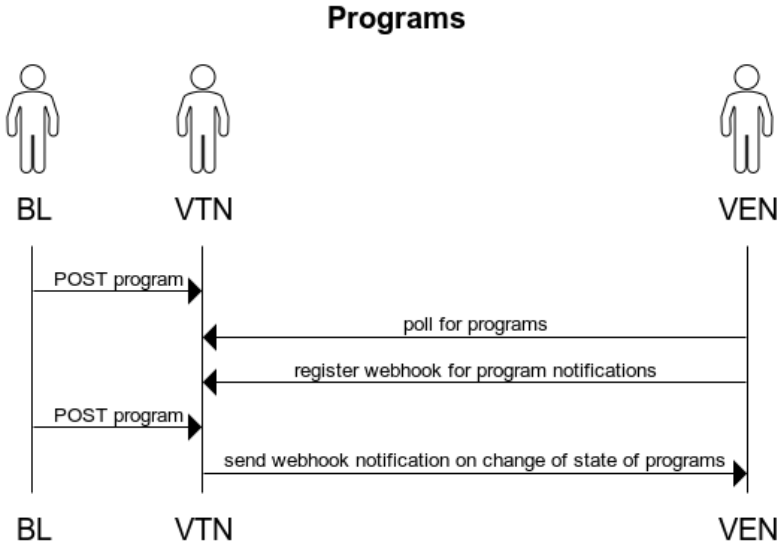



**Figure 4. Program Scenario**

### 6.5 Event Scenarios

The OpenADR framework supports a number of different Demand Response scenarios. A representative sample is documented here to ensure that OpenADR 3.0 fully supports a range of common use cases. For background see [OADR-2.0b-Program_Guide]**.**

- **Episodic Demand Response:** As BL I want to communicate a Demand Response signal to participating VENs. In this scenario, the event communicates a single start and duration pair. VENs are expected to be pre-programmed, according to an associated Program Description, to interpret the signal, manage associated resources, and generate a corresponding set of reports. Examples include Direct Load Control and Load Shed.
- **Continuous Demand Response.** As a VTN I want to communicate Dynamic Pricing signals to participating VENs. In this scenario, an event communicates a set of start and duration pairs with associated price values. A representative example is sending day-ahead variable pricing for each hour of the following day. VENs are expected to interpret the event(s) and to generate a corresponding set of reports. EV (Electric Vehicle), TOU (Time of Use), CPP (Critical Peak Price),

and VPP (Variable Peak Price) prices can be readily encoded as can more continuous prices such as hourly or smaller time periods.

BL will create events in the VTN, which VENs can query for, or receive notification of if they have previously registered a subscription. With day-ahead prices, they are fixed once announced and the intervals are never duplicated. With more real-time pricing, there may be a 24-hour forecast announced with each hour, but in subsequent hours, future prices may be adjusted as the forecast changes. It is assumed that a final price for each interval is announced before that interval begins.

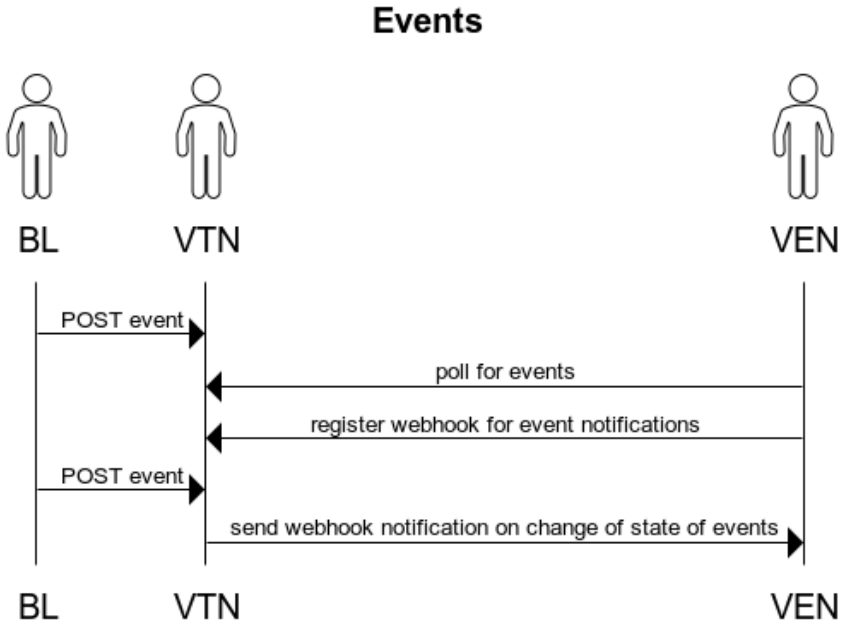



**Figure 5. Event Scenario**

### 6.6 Report Scenarios

OpenADR 3.0 defines mechanisms to support the most widely used types of reports. A catalog of enumerated report types (specified by report type) and qualifiers defined by [OADR-3.0-Enumerations] allow Business Logic clients to request reports via programs and events, with configuration details, and VENs to respond with the appropriate reports.

Reports may be requested by a program, by inclusion of one or more reportDescriptors, or may be requested by events. The data structures for events and reports are as identical as practical. This design allows VENs to determine how and when to send reports, and BL to unambiguously associate data included in reports to event elements. Example reports are:

- **Hourly prices and usage**
  An electricity retailer may publish day ahead prices for each hour of a day, and bill customers at those prices according to usage over each hour. In this scenario, it is critical that usage data corresponds exactly to the hourly rates, that is, a VEN reports data for each hour as described in the event intervals.

  BL may generate events that specify this behavior via a reportDescriptor with the appropriate enumerations. A VEN responds with appropriate reports.

- **Device status**
  BL requires the ability to poll resources enrolled in a program to determine their operational state. BL can generate an event containing a reportDescriptor that includes enumerations that indicate

it requires a near-real time response indicating the state of every resource under control of each VEN. An event can limit the scope of VENs and resources by providing a set of targets.

- **Load shed**
  BL generates an event that signals the start/end time of a load shed event, including report requirements via a list of reportDescriptors that may be interpreted by a VEN as a request to respond prior to the start of the event with a report indicating resource load shed capacity, and respond at the end of the event with an indication of amount of shed load for each resource.

  In response to report requests in events, VENs will create reports in the VTN, which Business Logic can query for or receive notification of via subscription.

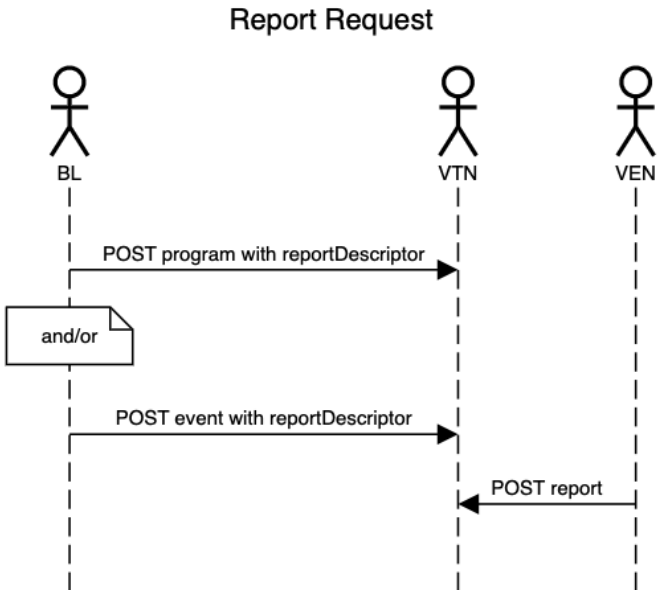



**Figure 6. Report Scenario**

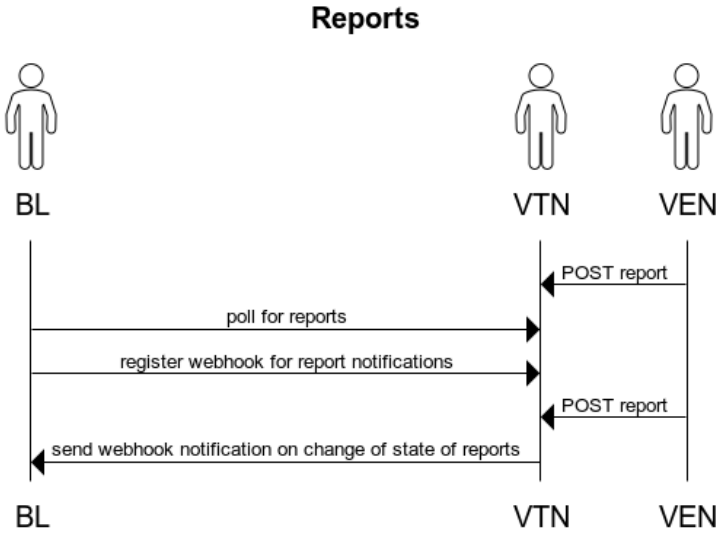



**Figure 7. Report Scenario**

### 6.7 VEN and Resource Scenarios

In order to assign VENs and resources to groups for the purposes of targeting, the API provides mechanisms for clients to create VEN objects, and VEN's may include a list of resource objects. Each of these objects contains a list of group labels that can be modified over time.

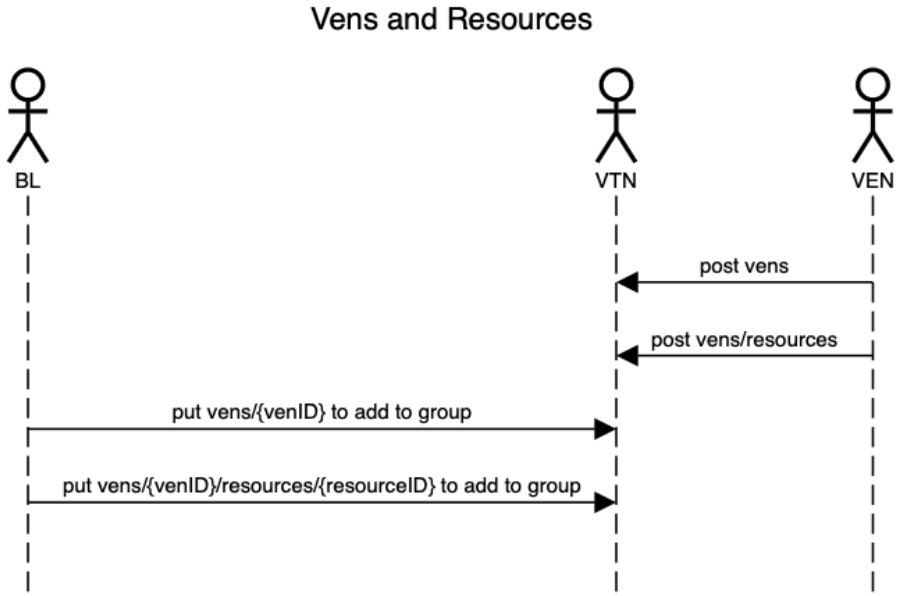



**Figure 8. VEN and Resource Scenario**

## 7 General Usage

### 7.1 Required and optional properties

If a representation sent to the VTN lacks a required property, the server returns a 400, Bad Request value. A resource is not created or updated.

If a representation does not include an optional (non-required) property, a corresponding object is created. Representations of the created object provided by the server will not include optional properties that have their default value.

If the representation includes a property that is not defined by the model, a corresponding resource is created without the additional property. That is, the additional content is ignored.

### 7.2 Response Codes and Errors

The API returns the following standard codes:

GET
- 200 - OK
- 400 - Bad Request
- 403 - Forbidden
- 500 - Internal Server Error

POST
- 201 - Created
- 400 - Bad Request
- 403 - Forbidden
- 409 - Conflict (item already exists)
- 500 - Internal Server Error
- 507 - Insufficient Storage

PUT
- 200 - OK
- 400 - Bad Request
- 403 - Forbidden
- 500 - Internal Server Error

DELETE
- 200 - OK
- 400 - Bad Request
- 403 - Forbidden
- 500 - Internal Server Error

On 40x and 500 responses, the API will provide a *problem* object that contains details of the error. The problem object will help clients determine what caused a 400 response, such as Bad Request, Unauthorized, Forbidden, etc.

### 7.3 POST and PUT

POST is used to create new objects, and PUT is used to update an existing object. objectID and createdDateTime values included in representations used in POST and PUT requests will be ignored by the VTN server.

### 7.4 API explorer

[OADR-3.0-Reference_Implementation] provides an online view of the OpenADR 3.0 OpenAPI specification and interactive features to view and explore every operation on every resource. One may also use tools provided by [swaggerhub] to view and explore the API.

### 7.5 Compression

In some circumstances the size of responses must be minimized, such as over bandwidth-constrained connections.

The HTTP protocol allows a client to request encoding using one of several compression algorithms, such as GZIP, by using the Accept-Encoding header. Server responses can then use the Content-Encoding header to indicate the algorithm chosen, and the response body is compressed accordingly. The result should be a seamless exchange where data is compressed.

Configuring this behavior is outside the scope of the OpenAPI definition of the OpenADR protocol. Instead, compression is implemented through configuration of the HTTP/REST framework used for OpenADR clients and servers.

Support for a given compression format is optional for VTNs, although gzip is encouraged.

## 8 Feature Examples

The following subsections illustrate some of the features of OpenADR 3.0. This is not exhaustive and other features are illustrated by specific use cases below.

JSON examples may represent something a client sends to the VTN or something returned by the server. In the former a client may send a minimal representation while the server will provide a more comprehensive representation that includes an object id, created and modified date stamps, and object type. All POST examples can be used in a request to a VTN.

### 8.1 Event Priority

Events can have a priority. This is used if there is the possibility of events overlapping and that they conflict (events can overlap and not conflict, e.g. for separate data streams, e.g. prices and GHG values, or prices and alerts). Lower numbers have a higher priority, and the highest priority is zero. If two events conflict and they have the same priority, the resulting behavior is not specified by OpenADR.

### 8.2 Object References

OpenADR 3.0 objects may make reference to other addressable objects. For example, an event object is required to include a reference to an associated program. An addressable object is one that can be accessed directly via the API and is populated with an objectID by the VTN on creation.

objectID references are explicit references to specific objects in the VTN. The objectID is independent of any user-assigned attribute of the program object.

The usage scenario is that when a client wishes to create object A that includes a reference to object B, it will first read object B to find its objectID, and use that to populate a reference attribute of object A.

For example, when creating an event, a BL client would read the associated program object and write its objectID into the event.programID attribute.

Note that direct object references are not the same as object names used in other contexts. For example, a report.resources.clientName represents an application-defined string and is not a direct reference to an existing resource object on the VTN server. In this case, reports may be created without the need to populate resource objects on the server, which exist primarily to support event targeting.

Events and reports are always associated with a program, and reports are also typically associated with events. These associations are maintained by making direct object reference. For example, an event will contain the object reference of the associated program:

POST minimal program

```
{
       "programName": "minimalProgram"
}
```

GET program

```
{
  "bindingEvents": false,
  "createdDateTime": "12:58:08",
  "id": "44",
  "localPrice": false,
  "objectType": "PROGRAM",
  "programName": "minimalProgram"
}
```

POST minimal event

```
{
  "eventName": "minimalEvent",
  "programID": "44",
  "intervalPeriod": {
```

```
    "start": "0"
  },
  "intervals": [{
    "id": 0,
    "payloads": [{
      "type": "PRICE",
      "values": [0.17]
    }]
  }]
}
```

GET event

```
{
  "createdDateTime": "13:21:51",
  "eventName": "minimalEvent",
  "id": "1",
  "intervalPeriod": {
    "start": "0"
  },
  "intervals": [
    {
      "id": 0,
      "payloads": [
        {
          "type": "PRICE",
          "values": [
            0.17
          ]
        }
      ]
    }
  ],
  "objectType": "EVENT",
  "programID": "44"
}
```

**Figure 9. Program and Event Examples.**

A report contains references to both program and event objects:

POST minimal report

```
{
  "reportName": "minimalReport",
```

```
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "resources": []
}
```


GET report


```
{
  "clientName": "myClient",
  "createdDateTime": "13:25:20",
  "eventID": "1",
  "id": "1",
  "objectType": "REPORT",
  "programID": "44",
  "reportName": "minimalReport",
  "resources": []
}
```


**Figure 10. Report Example**

### 8.3 Event and Interval Timing

OpenADR 2.0b introduced events and lists of intervals included in events. OpenADR 3.0 maintains these constructs but adapts them to fit the REST model.

Events typically include a list of one or more intervals, which define a temporal window, and include one or more payloads, such as a price value.

Reports are generally created by VENs in response to report requests and explicitly refer to the intervals within the associated event.

Intervals may be of even duration and contiguous. In this case, a single intervalPeriod can be included in an event object to set the beginning time of the first interval and the fixed duration of all intervals.

To create intervals of unequal duration, or to have them overlap or have temporal gaps, each interval can include its own intervalPeriod object. If an intervalPeriod is present in the event, it defines default values that are overridden by any intervalPeriods at the interval level.

The lifespan of an event is the start time of the first interval plus the sum of all interval durations, adjusted for overlaps or gaps introduced by start times in individual intervals.

An event on the server whose intervals have all transpired is effectively expired and can be deleted by BL.

In order to describe repeating intervals, a VTN or VEN may set the duration value on an intervalPeriod at the event level to -1. This indicates that event and report payloads associated with intervals are to repeat until the event is deleted. This requires that individual intervals must define their own durations, as there is no default. This feature facilitates such use cases as indefinite report streams, for example, recurring 5 minute usage readings.

### 8.4 Report Management

A reportDescriptor includes attributes to allow BL to control how reports are to be generated. Several examples are provided below to illustrate how these attributes are intended to be used.

Report requests are associated with an event via reportDescriptors. The intervals in the event are used to specify the timing of reporting; reporting attributes refer to intervals and not timing parameters.

The attributes used to control reporting are type, readingType, aggregate flag, and targets plus the following:

- startInterval
  - The interval (counted from the first, or zero-ith) at which to generate a report.
  - Default = -1. Indicates end of last interval.
- numIntervals
  - The number of intervals to include in a report
  - Default = -1. Indicates include all intervals.
- historical
  - The direction of intervals included in a report from the startInterval. If true, intervals precede the start to transmit historical data, if false the report includes only forward looking information, e.g. a forecast.
  - Default: true. Report includes historical information.
- frequency
  - The number of intervals between each report
  - Default: -1. Indicates all intervals, e.g. a single report
- repeat
  - Number of times to repeat. -1 indicates repeat indefinitely.

The following diagram provides a conceptual model of the reportDescriptor

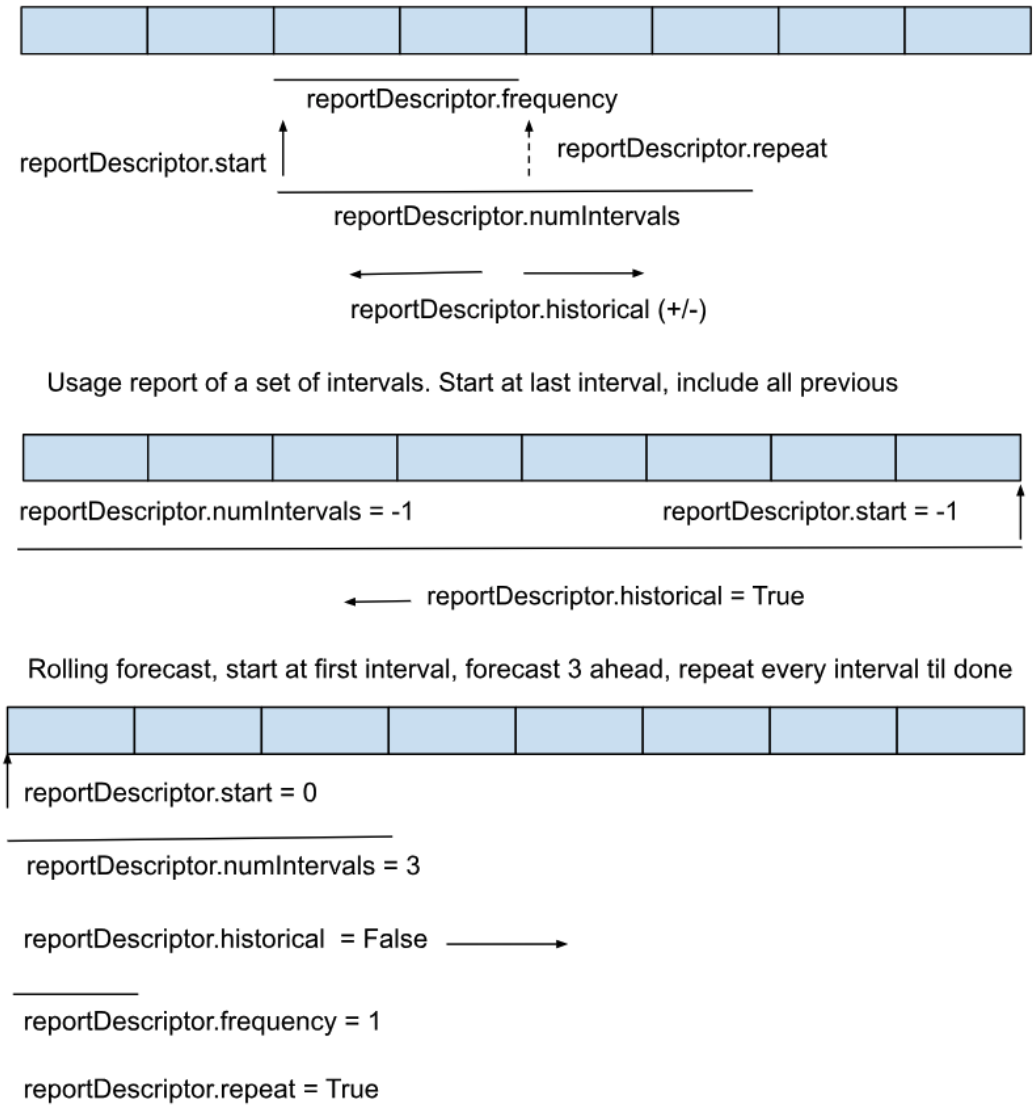



**Figure 11. reportDescriptor Details**

The use of an intervalPeriod at the event level with duration = -1 is a special case that denotes that the event's intervals continue into the future until the event is deleted. This is useful for requesting a continuous stream of reports for example. However, since duration in this case does not indicate the duration of individual intervals, each interval in the event's interval list must include an intervalPeriod definition with at least the duration field present. For example, in the case of a 48 hour rolling forecast, 48 intervals would be defined.

### 8.5 Variable duration intervals

Business logic clients have the option to attach an intervalPeriod to an event and to individual intervals. When only present in the event, the duration of all intervals are as specified in the event's intervalPeriod. An intervalPeriod in an interval will override what is specified at the event level, if present. If an intervalPeriod is not present in the event, then one must be present in every interval.

The example below shows an event with a set of intervals with the same duration, as specified by the intervalPeriod construct in the event. The second interval defines its own duration.

POST event

```
{
  "eventName": "variableIntervalsEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "2023-02-10T00:00:00.000Z",
     "duration": "PT1H"
  },
  "intervals": [{
        "id": 0,
        "payloads": [{
           "type": "PRICE",
           "values": [0.17]
        }]
     },
     {
        "id": 1,
        "intervalPeriod": {
           "start": "0",
           "duration": "PT2H"
        },
        "payloads": [{
           "type": "PRICE",
           "values": [0.22]
        }]
     }
  ]
}
```

**Figure 12. Variable Duration Interval Event**

### 8.6 payloadDescriptors

Events and reports generally contain one or more intervals, each of which may contain one or more payload objects. The payload object is purposely kept simple, just type and values attributes. The payloadDescriptor provides full context for a payload while avoiding duplicating static data across a potentially large series of intervals. For example, the values in payload with type of PRICE are simply numbers, and an accompanying payloadDescriptor supplies the units and currency information necessary to fully interpret the price values.

A payloadDescriptor contains a type attribute which associates it to payloads of the same type.

Note that events do not have a type. The type of an event is implicit in the type contents that it contains. For example, an event with prices and GHG values can be thought of as a price event, but in OpenADR 3.0, there is no manifestation of this other than the contents of payloadDescriptors. The same applies to reports.

POST event


```
{
  "eventName": "payloadDescriptorsEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "2023-02-10T00:00:00.000Z",
     "duration": "PT1H"
  },
  "payloadDescriptors": [{
        "payloadType": "PRICE",
        "units": "KWH",
        "currency": "USD"
     },
     {
        "payloadType": "GHG"
     }
  ],
  "intervals": [{
       "id": 0,
     "payloads": [{
           "type": "PRICE",
           "values": [
              0.17
           ]
        },
        {
           "type": "GHG",
           "values": [
              0.03
           ]
        }
     ]
  }]
}
```


**Figure 13. payloadDescriptors Event**

A payloadDescriptor list may also be included in a program object. This provides default values for all events associated with a program.

### 8.7 Aggregated Report

Where a VEN aggregates data from a number of resources, it may provide a single resource entry in the resources list and set the resourceName to AGGREGATED_REPORT. Aggregation means the data from a set of resources are summed.

#### 8.7.1 Event

The event in this example requests an aggregate report by setting reportDescriptors[0].aggregate = True

POST event

```
{
  "eventName": "reportDescriptorsEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "2023-02-10T00:00:00.000Z",
     "duration": "PT1H"
  },
  "reportDescriptors": [{
     "payLoadType": "USAGE",
     "readingType": "DIRECT_READ",
     "aggregate": "True"
  }],
  "intervals": [{
       "id": 0,
     "payloads": [{
           "type": "PRICE",
           "values": [
              0.17
           ]
        }
     ]
  }]
}
```

**Figure 14. Aggregated Report Request Event**

### 8.7.2 Report

GET report

```
{
  "reportName": "aggregatedReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "resources": [{
     "resourceName": "AGGREGATED_REPORT",
     "intervals": [{
        "id": 0,
        "payloads": [{
           "type": "USAGE",
           "values": [0.012]
        }]
     }]
  }]
}
```

**Figure 15. Aggregated Report Example**

### 8.8 Data Quality

When there is a need to characterize the data quality of a payload value, a payload of type DATA_QUALITY can be added to a payload array.

POST report

```
{
  "reportName": "DataQualityReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "resources": [{
     "resourceName": "AGGREGATED_REPORT",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
              "type": "USAGE",
              "values": [0.012]
           },
           {
              "type": "DATA_QUALITY",
              "values": ["MISSING"]
           }
        ]
     }]
  }]
}
```

**Figure 16. Data Quality Report**

This approach is limited to payload lists that would normally contain a single entry so that the data quality payload is unambiguously associated with that payload. For example, if the payload list contained payloads for usage and GHG data, there is currently not a way to describe the data quality of one specifically, or both. In such a case, one could create additional intervals, with identical intervalPeriods, for each payload type, or additional events.

### 8.9 Subscriptions

OpenADR 3.0 supports both PULL and PUSH interactions. PULL interactions are initiated by a client and are implemented with HTTP GET, POST, PUT, and DELETE.

Push interactions are implemented by Subscriptions. These allow a client to be notified when certain operations transpire and are implemented via webhooks. A webhook is a client endpoint registered with

the VTN. When a specified operation occurs, for example when a resource like an event is created, a request to the client endpoint is made by the server, thus notifying the client of a new event.

Webhooks are implemented using subscription objects. A client creates a new subscription object, which contains the webhook callback URL, and a description of the objects and operations that will trigger a request to the callback URL.

POST subscription

```
{
  "clientName": "myClient",
  "programID": "44",
  "objectOperations": [{
     "callbackUrl": "https://myserver.com/callbacks",
     "operations": [
        "POST",
        "PUT"
     ],
     "objects": [
        "EVENT",
        "PROGRAM"
     ]
  }]
}
```

**Figure 17. Subscription**

A subscription object is associated with the specific client and specific program. The resourceOperations list specifies what operations on which resource types the client requests to be notified about.

A client may provide multiple resourceOperation entries to provide different callbackUrls to catch notifications of different resources and operations.

POST subscription

```
{
  "clientName": "myClient",
  "programID": "44",
  "objectOperations": [
     {
     "callbackUrl": "https://myserver.com/event_callbacks",
     "operations": [
        "POST",
        "PUT"
     ],
     "objects": [
        "EVENT"
     ]
  },
```

```
    {
    "callbackUrl": "https://myserver.com/program_callbacks",
    "operations": [
       "POST",
       "PUT"
    ],
    "objects": [
       "PROGRAM"
    ]
  }
   ]
}
```

**Figure 18. multiple callback Subscription**

A VTN will make a request to the callback URL when the conditions are met per the resourceOperations registered subscriptions. The callback request body is a Notifications object that indicates the object type of the resource, the operation that provoked the request, and the relevant resource object.

Notification

```
{
  "objectType": "PROGRAM",
  "operation": "POST",
  "object": {
     "bindingEvents": false,
     "createdDateTime": "15:51:29",
     "id": "0",
     "localPrice": false,
     "objectType": "PROGRAM",
     "programName": "myProgram"
  }
}
```

**Figure 19. Notification Example**

### 8.10 Response Filtering

Programs may support large numbers of VENs and associated resources. In order to reduce potentially large responses that might include unnecessary object representations, clients may provide query params to allow VTN servers to filter results.

For example, when requesting a list of reports, a client may specify, through query params, that only reports associated with a specific program, and/or created by a specific client, be included in a response.

Targeting criteria can be used to filter responses to include only those objects that are targeted to specific VENs and resources.

Program, event, and subscriptions may contain lists of targets which indicate sets of vens and resources that are the intended recipients of these objects. Vens and resources may include their own targeting

criteria to associate themselves with targets. Clients may provide a list of targets when requesting a list of these objects to allow a VTN server to only respond with those objects that match the targeting criteria.

Filtering query params are applied to the following operations:

GET <>/programs: targets
GET <>/reports: programID, clientName
GET <>/events: programID, targets
GET <>/subscriptions: programID, clientName, targets
GET <>/vens: targets
GET <>/vens/{venID}/resources: targets

All GET operations allow clients to provide limit and skip query params to control pagination of potentially large responses.

Future versions of OpenADR may provide other features that facilitate massive scalability or operation under extremely bandwidth or memory constrained environments.

### 8.11 Dynamic Targeting

For programs that require that VENs or their resources be assigned targeting labels at runtime, the API provides representations of VENs and resources with targetValues as attributes.

VENs may create a VEN object to represent themselves, and each VEN object may be populated with a list of resource objects. BL may update these objects with targetValues, and subsequently target events and report requests to these target strings.

A VEN may create a VEN object by POSTing a VEN representation to <>/vens/

POST ven

```
{
  "venName": "VENID_0999"
}
```

**Figure 20. Dynamic Targeting Ven**

The venName may have been provisioned by BL during an out-of-band process.

To create resources, a VEN may POST to <>/vens/{venID}/resource

POST resource

```
{
  "resourceName": "RESOURCE_0999"
}
```

**Figure 21. Dynamic Targeting Resource**

A GET of <>/vens/{ven.id} would yield:

```
{
       "createdDateTime": "15:55:25",
       "id": "0",
       "objectType": "VEN",
       "resources": [{
              "createdDateTime": "15:59:41",
              "id": "0",
              "objectType": "RESOURCE",
              "resourceName": "RESOURCE_0999",
              "venID": "0"
       }],
       "venName": "VENID_0999"
}
```


**Figure 22. Another Dynamic Targeting Ven**

To manage groups and locations, clients can modify the targetValues attributes of VENs or resources via PUT, e.g.
PUT <>/vens/{ven.id}/resources/{resource.id}

PUT resource


```
{
  "resourceName": "RESOURCE_0999",
  "targets": [{
     "type": "GROUP",
     "values": ["group1"]
  }]
}
```


**Figure 23. Another Dynamic Targeting Resource**

### 8.12 Problem

When something goes wrong and a VTN responds to a request with an HTTP 4xx or 5xx status code, it will also return a problem object to provide error details.

For example:

problem


```
{
  "title": "Service Unavailable",
  "status": 503,
  "detail": "Database not reachable"
}
```


**Figure 24. Problem Example**

# 9 Use Cases

This section describes demand response Use Cases that are specifically supported by OpenADR. These are not formally defined as some Use Case templates require, but are intended as informative material for readers. This is not an exhaustive list as the protocol is flexible to accommodate many other uses than described here.

## 9.1 Alert

An Alert is an asynchronous message that generally occurs between a few times per year to only every few years. They are non-financial (do not themselves indicate a change in the customer's bill) and are most commonly used for some sort of emergency or other anomalous condition. BL generates alerts as it deems fit. As they are asynchronous, they are well suited to subscriptions rather than polling.

An Alert is of a type (e.g. 'Grid Emergency') suitable for use by devices, and a free text string intended to be shared with any human involved with the customer site. For many alerts, the distribution system may be damaged interrupting power supply, and devices and humans may change their operation in response to this and other impacts of the alerts.

POST event

```
{
  "eventName": "alertEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "0"
  },
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "ALERT_GRID_EMERGENCY",
        "values": ["The grid is currently under emergency conditions"]
     }]
  }]
}
```

**Figure 25. Alert Event**

## 9.2 Load Shed

A BL logic entity wishes to provide ad-hoc events to VENs to signal resources of an upcoming Load Shed 'event'. Traditionally, resources have enrolled in an energy retailer's load shed offering and await a signal indicating a temporal window during which to modify energy consumption. Specific examples following this pattern are critical peak prices, direct load control, and simple shed events, as from [OADR-2.0b-Program_Guide].

### 9.2.1 Critical Peak Pricing Program (CPP, VPP)

This use case employs the SIMPLE event type to communicate Load Shed windows based on high wholesale market prices or power system emergency conditions. This can be extended to a Variable Peak Pricing Program (VPP) of up to three different high prices with the SIMPLE event type.

### 9.2.2 Direct Load Control/Thermostat Program

A Demand Response event directly modifies the behavior of load shedding resources, without a layer of abstraction between receipt of the signal and the specific load shedding action taken. Direct Load Control is not typically implemented using OpenADR, but this use case is included as an example of how it might be used in the case of a Thermostat.

### 9.2.3 Events

As indicated in the payloadDescriptors list, events contain payloads of type SIMPLE. A SIMPLE payload contains a value of 0, 1, 2, or 3 indicating a resource-defined load shed level. In this example, the event contains an intervalPeriod with a specific start time and a duration of 4 hours.

POST event

```
{
  "eventName": "simpleEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "0"
  },
  "payloadDescriptors": [{
     "payloadType": "SIMPLE"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "SIMPLE",
        "values": [1]
     }]
  }]
}
```

**Figure 26. SIMPLE Event**

### 9.2.4 Reports

No reports are required for this basic scenario.

## 9.3 Day Ahead Prices with Usage Report

Energy retailers may wish to provide dynamic pricing to encourage load shaping by flexible load resources. In other words, a fluctuating price encourages resources to consume more during periods of relatively low prices and less in the high price periods.

### 9.3.1 Program

The configuration of a program object may contain a payloadDescriptors list indicating the event type PRICE and report type USAGE. Note that payload type values do not indicate whether they are event or report payloads as the definition of the value fully describes their usage. Note also the inclusion of ‘units’ and ‘currency’ attributes; these are included in order for consumers of PRICE events to fully interpret a price value.

POST program

```
{
  "programName": "payloadDescriptorsProgram",
  "payloadDescriptors": [{
     "payloadType": "PRICE",
     "units": "KWH",
     "currency": "USD"
  }]
}
```

**Figure 27. Price Program**

### 9.3.2 Events

As indicated in the program payloadDescriptors list, events are expected to contain payloads of type PRICE.

A PRICE payload contains a float value. In the example, the event contains an intervalPeriod with a specific start time and a duration of 1 hour. This indicates that every interval spans 1 hour, therefore 24 intervals describe 24 hours.

This event contains a reportDescriptor that indicates the VEN is expected to create a USAGE report of reading type DIRECT_READ once the last interval has transpired.

POST event

```
{
  "eventName": "pricingEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "2023-02-10T00:00:00.000Z",
     "duration": "PT1H"
  },
  "reportDescriptors": [{
     "payloadType": "USAGE",
     "readingType": "DIRECT_READ",
     "startInterval": -1,
     "numIntervals": -1,
     "historical": true,
     "frequency": -1,
     "repeat": 1
  }],
  "payloadDescriptors": [{
        "payloadType": "PRICE",
        "units": "KWH",
        "currency": "USD"
     }
  ],
  "intervals": [{
     "id": 0,
     "payloads": [{
```

```
            "type": "PRICE",
            "values": [
               0.17
            ]
         },
         {
            "type": "PRICE",
            "values": [
               0.03
            ]
         }
      ]
   }]
}
```

**Figure 28. Usage Report Request Event**

### 9.3.3 Reports

As indicated in the event reportDescriptor, the VEN will generate a USAGE report when the last interval has transpired. A USAGE report is expected to include an identical number of intervals, with identical IDs, start times, and durations, as the associated event.

"startInterval": -1 (default) indicates that a report should be generated after the last interval has transpired.

"numIntervals": -1 (default) indicates the report should include readings for all intervals.

"historical": True (default) indicates that reporting should include the intervals that precede the startInterval, otherwise the report would be future-looking.

"frequency": -1 (default) indicates that the report should be generated after all intervals have transpired.

"repeat": 1 (default) indicates only one report is to be generated.

The above values are all the defaults, so an equivalent reportDescriptor is:

reportDescriptor

```
{
    "payloadType": "USAGE",
    "readingType": "DIRECT_READ"
 }
```

**Figure 29. reportDescriptor snippet**

The resulting report indicates the program and event it is associated with. The report includes a clientID. Presumably this value is assigned to a VEN via an out-of-band registration flow.

The report contains a list of resource objects to provide usage data for separate resources under control of the VEN.

POST report

```
{
  "reportName": "usageReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "payloadDescriptors": [{
     "payloadType": "USAGE",
     "readingType": "DIRECT_READ",
     "units": "KWH"
  }],
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
              "type": "USAGE",
              "values": [0.012]
           },
           {
              "type": "USAGE",
              "values": [0.017]
           }
        ]
     }]
  }]
}
```

**Figure 30. Usage Report**

## 9.4 Inverter Management

Energy retailers may wish to coordinate inverters (or other devices) to adjust their behavior over time based on fluctuating values, such as Volt-VAR settings.

### 9.4.1 Program

A program object is expected to contain a payloadDescriptors list indicating the event type CURVE. Other attributes of the program are program-specific.

POST program

```
{
  "programName": "inverterProgram",
  "payloadDescriptors": [{
```

```
    "payloadType": "CURVE"
  }]
}
```

**Figure 31. Inverter Management Program**

### 9.4.2 Events

As indicated in the program payloadDescriptors list, events are expected to contain payloads of type CURVE. A CURVE payload contains a list of float values. In this example, the event contains an intervalPeriod with a specific start time and a duration of 1 hour.

This event example does not contain a reportDescriptor, therefore a VEN will not be expected to generate a report based on resource behavior in response to this event.

POST event

```
{
  "eventName": "inverterEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT1H"
  },
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "CURVE",
        "values": [{
              "x": 0.17,
              "y": 0.26
           },
           {
              "x": 0.19,
              "y": 0.28
           }
        ]
     }]
  }]
}
```

**Figure 32. Inverter Management Event**

### 9.4.3 Reports

None required in this example

## 9.5 Load Control

Send day ahead LOAD_DISPATCH event signal targeted to one or many aggregators, with the aggregator managing the load profile of behind the meter batteries, water heaters, ‘BYOD’ thermostats, and EV.

POST event

```
{
  "eventName": "loadControlEvent",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT4H"
  },
  "payloadDescriptors": [{
     "payloadType": "DISPATCH_SETPOINT",
     "units": "KW"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "DISPATCH_SETPOINT",
        "values": [0]
     }]
  }]
}
```

**Figure 33. Load Control Event**

No reporting example.

### 9.6 State of Charge Reporting

A grid entity seeks to have situational awareness of the energy state of storage based devices such as Electric Vehicles and Battery Storage. In this example, an event is used to request information about storage resources.

#### 9.6.1 Event

```
{
  "name": "SOC_report_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "reportDescriptors": [{
        "payLoadType": "STORAGE_USABLE_CAPACITY"
     },
     {
        "payLoadType": "STORAGE_CHARGE_LEVEL"
     },
     {
```

```
        "payLoadType": "STORAGE_MAX_DISCHARGE_POWER"
     },
     {
        "payLoadType": "STORAGE_MAX_CHARGE_POWER"
     }
  ],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "REQUIRED_BUT_NOT_USED",
        "values": [0]
     }]
  }]
}
```

**Figure 34. State of Charge Reporting Event**

### 9.6.2 Report

POST report

```
{
  "reportName": "SOCReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "payloadDescriptors": [{
        "payLoadType": "STORAGE_USABLE_CAPACITY",
        "units": "KWH"
     }, {
        "payLoadType": "STORAGE_CHARGE_LEVEL",
        "units": "PERCENTAGE"
     }, {
        "payLoadType": "STORAGE_MAX_DISCHARGE_POWER",
        "units": "KWH"
     }, {
        "payLoadType": "STORAGE_MAX_CHARGE_POWER",
        "units": "KWH"
     }],
     "intervals": [{
        "id": 0,
```

```
        "payloads": [{
              "type": "STORAGE_USABLE_CAPACITY",
              "values": [0.018]
           },
           {
              "type": "STORAGE_CHARGE_LEVEL",
              "values": [33]
           },
           {
              "type": "STORAGE_MAX_DISCHARGE_POWER",
              "values": [0.011]
           },
           {
              "type": "STORAGE_MAX_CHARGE_POWER",
              "values": [0.007]
           }
        ]
     }]
  }]
}
```

**Figure 35. State of Charge Report**

### 9.7 Capability Forecast Reporting

Receive a rolling forecast of the aggregated load flexibility so that the BL is aware of how much load shed or generation can be dispatched up to 48 hours in the future.

#### 9.7.1 Event

Post event

```
{
  "eventName": "capability_report_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "-1"
  },
  "reportDescriptors": [{
        "payloadType": "LOAD_SHED_DELTA_AVAILABLE",
        "startInterval": 0,
        "numIntervals": 48,
        "historical": false,
        "frequency": 1,
        "repeat": -1
     },
     {
        "payloadType": "GENERATION_DELTA_AVAILABLE",
```

```
        "startInterval": 0,
        "numIntervals": 48,
        "historical": false,
        "frequency": 1,
        "repeat": -1
     }
  ],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "REQUIRED_BUT_NOT_USED",
        "values": [0]
     }]
  }]

}
```

**Figure 36. Capability Forecast Reporting Event**

### 9.7.2 Report

POST report

```
{
  "reportName": "capabilityReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "payloadDescriptors": [{
     "payloadType": "LOAD_SHED_DELTA_AVAILABLE",
     "units": "KWH"
  }, {
     "payloadType": "GENERATION_DELTA_AVAILABLE",
     "units": "PERCENTAGE"
  }],
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
              "type": "LOAD_SHED_DELTA_AVAILABLE",
              "values": [30.0]
           },
```

```
            {
               "type": "GENERATION_DELTA_AVAILABLE",
               "values": [110.0]
            }
         ]
      }]
   }]
}
```

**Figure 37. Capability Forecast Report**

### 9.8 Operational Forecast Reporting

Receive a forecast of the aggregated resource load utilization or generation taking into account planned price and event optimizations and other operational considerations so that the upstream VTN's application layer is aware of how much load utilization is likely to occur over the reporting period.

POST event

```
{
  "eventName": "ops_forecast_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "-1"
  },
  "reportDescriptors": [{
     "payloadType": "USAGE",
     "aggregate": true,
     "startInterval": 0,
     "numIntervals": 24,
     "historical": false,
     "frequency": 1,
     "repeat": -1
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "REQUIRED_BUT_NOT_USED",
        "values": [0]
     }]
  }]
}
```

**Figure 38. Operational Forecast Reporting Event**

The reportDescriptors in the example specifies an indefinite series of hourly forecasts for the following 24 hours, available load shed and generation.

Note the event's intervalPeriod.duration = -1. This requires that each interval define its own duration.

"startInterval": 0 indicates that a report should be generated when the first interval has begun (see historical = False).

"numIntervals": 24 indicates the report should include readings for 24 intervals

"historical": False indicates that reporting should include the intervals that follow the startInterval.

"frequency": 1 indicates that a report should be generated after every interval

"repeat": -1 indicates that reports will repeat until the last interval has transpired, which is not defined as the event duration is -1. Repeat automatically increments the virtual startInterval by frequency amount.

POST report

```
{
  "reportName": "opsForecastReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "payloadDescriptors": [{
     "payloadType": "USAGE",
     "units": "KW"
  }],
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
              "type": "USAGE",
              "values": [0.04]
           }

        ]
     }]
  }]
}
```

**Figure 39. Operational Forecast Report**

### 9.9 2.0b Program Guide Use cases

The following use cases are detailed in the [OADR-2.0b-Program_Guide] and are referenced here to ensure OpenADR 3.0 has feature parity with 2.0b.

*Italicized* text are paraphrased quotes of the descriptions of events and reports from [OpenADR-2.0b-PG] . Where SIMPLE signals are described, this generally indicates compatibility with OpenADR 2.0a.

### 9.9.1 Critical Peak Pricing

*Event Signals: If the deployment supports B profile VENs, in addition to the SIMPLE signal, an ELECTRICITY_PRICE signal may be included in the payload with a type of priceRelative, priceAbsolute, or priceMultiplier depending on the nature of the program.*

*Reporting Services: Telemetry reporting is typically not used as it is not absolutely necessary for CPP programs*.

This Use Case is addressed above in the section titled “Day Ahead Prices with Usage report”. Note that prices in OpenADR 3.0 are always absolute and the type is simply PRICE.

### 9.9.2 Thermostat Program

*Event Signals: A LOAD_CONTROL signal may be included in the payload with a type of x-loadControlLevelOffset or x-loadControlCapacity to specify the desired temperature setpoint offset or thermostatic cycling percentage respectively. It is recommended that a unit type of "temperature" be used in payloads utilizing the x- loadControlLevelOffset signalType to indicate Celsius or Fahrenheit for the offset.*

*Reporting Services: Telemetry status reports for small commercial Thermostat programs may be required, reporting at a minimum current setpoint offset of the thermostats which control the load shedding resources, as well as online/override status.*

This Use Case is partially addressed above in the section titled “Load Shed”**.** Detailed reporting items are not directly supported. OpenADR 3.0 has a type of CONTROL_SETPOINT but does not support a thermostatic cycling percentage.

### 9.9.3 Fast DR Dispatch

*Event Signals: A dispatch in the form of a LOAD_DISPATCH signal may be included in the payload with signal types of setpoint or delta, and units of powerReal. This signal represents the desired “operating point” of the load and can be expressed either as an absolute amount of mW (i.e. setpoint) or some relative number of mW (i.e. delta) from the resource's current operating point.*

*Reporting Services: In some cases the telemetry may include other data points such as voltage readings and charge state (i.e. energy) in the case where the resource is some form of storage. In some cases the reporting frequency may be as high as every 2 seconds.*

These are two use cases; one to set the absolute setpoint of a resource with DISPATCH_SETPOINT and one to set a relative setpoint with CONTROL_SETPOINT. Each is illustrated below:

POST event

```
{
  "eventName": "fast_dr_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
```

```
  "payloadDescriptors": [{
     "payloadType": "DISPATCH_SETPOINT",
     "units": "KWH"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "DISPATCH_SETPOINT",
        "values": [0.50]
     }]
  }]
}
```

**Figure 40. Dispatch Setpoint Event**

The following example illustrates modifying a setpoint to a value below the present setpoint.

POST event

```
{
  "eventName": "setpoint_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "payloadDescriptors": [{
     "payloadType": "CONTROL_SETPOINT",
     "units": "KWH"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "CONTROL_SETPOINT",
        "values": [-0.15]
     }]
  }]
}
```

**Figure 41. Another Dispatch Setpoint Event**

There are also two distinct reporting scenarios. The first requires voltage readings. An event may request such a report via:

POST event

```
{
  "name": "setpoint_Event",
```

```
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "reportDescriptors": [{
     "payloadType": "SETPOINT",
     "readingType": "DIRECT_READ",
     "units": "VOLTAGE"
  }],
  "payloadDescriptors": [{
     "payloadType": "CONTROL_SETPOINT",
     "units": "KWH"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "CONTROL_SETPOINT",
        "values": [-0.15]
     }]
  }]
}
```

**Figure 42. Dispatch Setpoint Report Request Event**

With the resulting report:

POST report

```
{
  "reportName": "setpointReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "payloadDescriptors": [{
     "payloadType": "SETPOINT",
     "units": "PERCENT"
  }],
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
```

```
            "type": "SETPOINT",
            "values": [0.10]
         }

      ]
   }]
}]
}
```

**Figure 43. Another Dispatch Setpoint Report**

### 9.9.4 Residential EV Time of Use

*Event Signals: ELECTRICITY_PRICE signals*

*Reporting Services: No reporting needed, all data can come from the meter.*

This Use Case is addressed above in the section titled “Day Ahead Prices with Usage report”.

### 9.9.5 Public Station EV Real-Time Pricing

*Event Signals: ELECTRICITY_PRICE signals with prices.*

*Reporting Services: No reporting needed, but can be used if desired.*

This Use Case is addressed above in the section titled “Day Ahead Prices with Usage report”.

### 9.9.6 DER DR

*Event Signals: ELECTRICITY_PRICE signals with 24 one hour intervals of prices over a 24 hour period.*

*Reporting Services: No reporting needed*

This Use Case is addressed above in the section titled “Day Ahead Prices with Usage report”.

## 9.10 Capacity Management

Until recently, electric utility practice has been to allow any customer to import or export power at a level up to the line rating of the customer service at any time. This was not burdensome as the times when individual customers imported or exported at anomalously high levels were infrequent and not coordinated. The advent of excess PV export and vehicle charging have both changed that assumption. The options now available are to deny new customer DER connections (PV and EV) for the possibility of worst case scenarios violating limits, reducing customer capacity below their line limits – or, introducing digital management of capacity.

How best to manage capacity is still an open question and can be expected to evolve in the coming years, but two mechanisms are included – one based on announcing limits and the other based on subscriptions for lower levels and permission-based increases in that level.

### 9.10.1 Dynamic Operating Envelopes

Dynamic Operating Envelopes (DOE) are an alternative mechanism for managing site capacity limits implemented in some jurisdictions (see DER Management Envelope as per Section 9 title in CSIP-AUS (SA HB 218:2023) SA HB 218:2023 | Standards Australia). They communicate a schedule of available

site-level import and/or export capacity, the operating envelope. The duration and interval length of the schedule will vary by application. When a new flexible load, such as an EV or stationary battery, wants to import power, it must restrict its consumption so that it does not contribute to an exceedance of the customer IMPORT_CAPACITY_LIMIT. Conversely for distributed generation, such as rooftop PV, that wants to export power, it must restrict its generation so that it does not contribute to an exceedance of the EXPORT_CAPACITY_LIMIT.

When a VEN connects to a VTN, the VTN will announce the customer's dynamic operating envelope schedule with a payload element of IMPORT_CAPACITY_LIMIT and/or EXPORT_CAPACITY_LIMIT if applicable), in units of kW, with an event interval commonly days-long. A new dynamic operating envelope schedule may be announced at any time, including many times per day.

A common usage is for each event to request a report on site operational parameters such as voltage, site-level active and reactive power demand as shown in the example below.

Post event


```
{
  "eventName": "capacity_limit_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "2023-02-10T00:00:00.000Z",
     "duration": "PT30M"
  },
  "reportDescriptors": [{
     "payloadType": "READING",
     "readingType": "DIRECT_READ",
     "units": "volts",
     "startInterval": -1,
     "numIntervals": -1,
     "historical": true,
     "frequency": -1,
     "repeat": 0
  }, {
     "payloadType": "READING",
     "readingType": "DIRECT_READ",
     "units": "kW",
     "startInterval": -1,
     "numIntervals": -1,
     "historical": true,
     "frequency": -1,
     "repeat": 0
  }, {
     "payloadType": "READING",
     "readingType": "DIRECT_READ",
     "units": "kVAr",
     "startInterval": -1,
     "numIntervals": -1,
```

```
    "historical": true,
    "frequency": -1,
    "repeat": 0
}],
"payloadDescriptors": [{
      "payloadType": "IMPORT_CAPACITY_LIMIT",
      "units": "KW"
   },
   {
      "payloadType": "EXPORT_CAPACITY_LIMIT",
      "units": "KW"
   }
],
"intervals": [{
      "id": 0,
      "payloads": [{
            "type": "IMPORT_CAPACITY_LIMIT",
            "values": [
               10.0
            ]
         },
         {
            "type": "EXPORT_CAPACITY_LIMIT",
            "values": [
               4.0
            ]
         }

      ]
   },
   {
      "id": 1,
      "payloads": [{
            "type": "IMPORT_CAPACITY_LIMIT",
            "values": [
               8.0
            ]

         },
         {
            "type": "EXPORT_CAPACITY_LIMIT",
            "values": [
               6.0
            ]
         }

      ]
```

```
    }
  ]
}
```

**Figure 44. Dynamic Operating Envelope with Site Parameters Event**

The above example illustrates 2 contiguous 30 minute intervals; additional intervals would cover additional time, e.g 48 30 minute intervals would define limits for an entire day.

Should an extended communications interruption occur and the scheduled IMPORT/EXPORT_CAPACITY_LIMIT enumeration values be exhausted, the VEN will utilise the IMPORT/EXPORT_CAPACITY_SUBSCRIPTION enumeration values as default limits. This would typically be announced from the VTN to the VEN on startup, with an event interval that could be years-long, and can be re-announced as needed with a new interval. Following a grid power outage, the VEN must disregard all existing IMPORT/EXPORT_CAPACITY_LIMIT, revert to the IMPORT/EXPORT_CAPACITY_SUBSCRIPTION and re-poll the VTN to establish if an updated IMPORT/EXPORT_CAPACITY_LIMIT schedule has been published.

### 9.10.2 Dynamic Capacity Management

Dynamic Capacity Management is a mechanism for addressing capacity constraints in the distribution system by creating a permission-based system of limiting customer power capacity. In a basic use case, customers subscribe to a capacity level that they normally never exceed for their ‘traditional’ end uses. When a new device, such as an EV or stationary battery, wants to import power, it can always do so at the difference between the rate of the *import subscription* level and the consumption of the balance of the system; to import power at a higher rate, a *import reservation* is required. Conversely for distributed generation, such as rooftop PV, that wants to export power, it can always generate at the total of the *export subscription* level and the consumption of the balance of the system; to export power at a higher rate, an *export reservation* is required. In both cases the coordination is with the customer site as a whole.

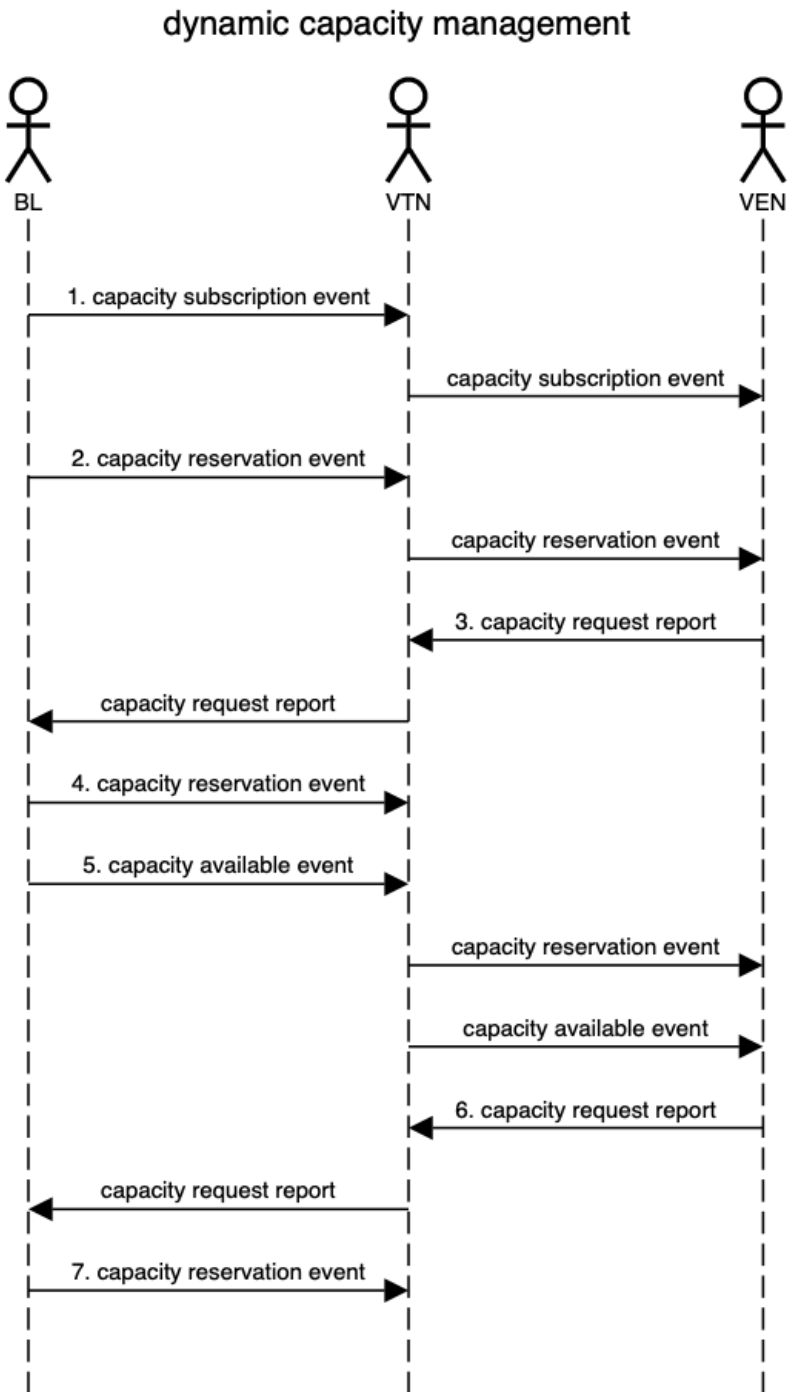
When a VEN connects to a VTN, the VTN will announce the customer’s subscribed capacity with a payload element of IMPORT_CAPACITY_SUBSCRIPTION (and/or EXPORT_CAPACITY_SUBSCRIPTION if applicable), in units of kW, with an event interval commonly years-long. If this is changed (through a process out of band of OpenADR) then a new subscription level is announced. The subscription should also include a report request that the VEN uses to request reservations.

At any time the customer may submit a *request* with a report including payload elements of IMPORT_CAPACITY_RESERVATION and IMPORT_CAPACITY_RESERVATION_FEE (and/or EXPORT_CAPACITY_RESERVATION and EXPORT_CAPACITY_RESERVATION_FEE), in units of kW and currency/kW (e.g. $/kW) respectively; this report can have multiple intervals, each of which generates an independent request. This is the additional capacity requested and the fee the customer is willing to pay. The customer will commonly begin with a fee of zero. The VTN will generate an event based on this to announce the customer’s upcoming reservations -- a combination of any past reservations that have not expired and successful new ones. To be a successful request, the requested capacity must be available and if the fee request must be sufficient. If multiple requests are submitted simultaneously, it is possible that some will be successful and some not. If the VEN submits a request with a fee higher than is required, the reservation fee is only the required amount. The upcoming reservations are announced with the IMPORT_CAPACITY_RESERVATION and IMPORT_CAPACITY_RESERVATION_FEE payload elements, similarly announced in units of kW and currency/kW). Finally, at the same time the VTN will send out an announcement of future capacity that can be reserved with the IMPORT_CAPACITY_AVAILABLE and IMPORT_CAPACITY_AVAILABLE_FEE payload elements with the same units. All of these four payload elements have counterparts for export which can be used in combination with or instead of import payloads. The VTN may reject reservations if the timing intervals do

not match what the VTN expects. For example, the VTN may expect reservations to be on 15-minute intervals and so reject ones that have start times or durations that don’t match this.

All of these are announced as intervals and can have one or many intervals in the sequence. Because the capacity information is customer-specific, this is a program requiring authentication, not a tariff relationship as the underlying energy prices are for the same customer.

The diagram below illustrates an example high-level interaction sequence, with a detailed description following.




The anticipated flow is (only numbered interactions are described):

1. BL logic sends a capacity subscription event to the VTN that contains subscription information. This event could include a report request for capacity request from VENs. A capacity subscription event includes a payload that contains the import and/or export capacity subscription (in kW) for the subscription.

2. BL may send a capacity reservation event to the VTN at any time. If the capacity subscription event did not include a capacity request report request, this event is expected to simply request a capacity request report from VENs.
3. When a VEN wishes to consume more power than its subscription level, it will send a capacity request report to the VTN. The report includes payloads that indicate the interval(s) over which extra capacity is requested and a powerLevel for each.
4. If BL reads a capacity request report it may create a new capacity reservation event targeted to the requesting VEN. This event is expected to include payloads that indicate a powerLevel over some set of intervals, thus providing a history of the reservations granted to the VEN. These results indicate to the VEN whether the requested reservation has been granted or not.
5. BL may also target a capacity available event to a VEN to indicate intervals over which capacity reservations might be requested. Such events include payloads containing powerLevel and price.
6. A VEN may generate a new capacity reservation request report in response to information gleaned from the previous capacity reservation event and capacity available event.
7. BL may grant or deny the latest reservation request from a VEN by targeting a capacity reservation event to a VEN..

Example request bodies for the above events and reports are as follows; all of the examples are for import but apply equally to export:

POST event

```
{
  "eventName": "capacity_subscription_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "payloadDescriptors": [{
     "payloadType": "IMPORT_CAPACITY_SUBSCRIPTION",
     "units": "KW"
  }],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "IMPORT_CAPACITY_SUBSCRIPTION",
        "values": [240]
     }]
  }]
}
```

**Figure 45. capacity subscription event**

POST event

```
{
  "eventName": "capacity_reservation_reportReq_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "reportDescriptors": [
   {"payloadType": "IMPORT_CAPACITY_RESERVATION"}
 ],
  "intervals": [{
     "id": 0,
     "payloads": [{
        "type": "REQUIRED_BUT_NOT_USED",
        "values": [0]
     }]
  }]
}
```

**Figure 46. Capacity reservation request event, report request only**

POST report

```
{
  "reportName": "capacitySubscriptionReport",
  "programID": "44",
  "eventID": "1",
  "clientName": "myClient",
  "payloadDescriptors": [{
     "payloadType": "IMPORT_CAPACITY_RESERVATION",
     "units": "KWH"
  }],
  "resources": [{
     "resourceName": "Resource_1",
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "intervals": [{
        "id": 0,
        "payloads": [{
              "type": "IMPORT_CAPACITY_RESERVATION",
              "values": [242]
           }
```

```
        ]
     }]
  }]
}
```

**Figure 47. capacity subscription report**

POST event

```
{
  "eventName": "capacity_reservation_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "reportDescriptors": [{
     "payloadType": "IMPORT_CAPACITY_RESERVATION"
  }],
  "payloadDescriptors": [{
        "payloadType": "IMPORT_CAPACITY_RESERVATION",
        "units": "KW"
     },
     {
        "payloadType": "IMPORT_CAPACITY_RESERVATION_FEE",
        "units": "USD"
     }
  ],
  "intervals": [{
     "id": 0,
     "intervalPeriod": {
        "start": "0",
        "duration": "PT0H"
     },
     "payloads": [{
           "type": "IMPORT_CAPACITY_RESERVATION",
           "values": [242]
        },
        {
           "type": "IMPORT_CAPACITY_RESERVATION_FEE",
           "values": [0.11]
        }
     ]
  }],
  "targets": [{
     "type": "VEN_ID",
     "values": ["VEN-99"]
  }]
```

```
}
```

**Figure 48. capacity reservation event**

POST event

```
{
  "eventName": "capacity_available_Event",
  "programID": "44",
  "intervalPeriod": {
     "start": "0",
     "duration": "PT0H"
  },
  "reportDescriptors": [{
     "payloadType": "IMPORT_CAPACITY_RESERVATION"
  }],
  "payloadDescriptors": [{
        "payloadType": "IMPORT_CAPACITY_AVAILABLE",
        "units": "KW"
     },
     {
        "payloadType": "IMPORT_CAPACITY_AVAILABLE_FEE",
        "units": "USD"
     }
  ],
  "intervals": [{
     "id": 0,
     "intervalPeriod": {
        "start": "2023-02-10T00:00:00.000Z",
        "duration": "PT1H"
     },
     "payloads": [{
           "type": "IMPORT_CAPACITY_AVAILABLE",
           "values": [242]
        },
        {
           "type": "IMPORT_CAPACITY_AVAILABLE_FEE",
           "values": [0.11]
        }
     ]
  }],
  "targets": [{
     "type": "VEN_ID",
     "values": ["VEN-99"]
  }]

}
```

**Figure 49. capacity available event**

### 9.11 OpenADR and CTA-2045

OpenADR 3.0 is well suited to be a standard external protocol for CTA-2045B, which is a standard for an external module on a flexible load or other device to provide replaceable interface devices with different communication and computation capabilities. For many capabilities of CTA-2045, e.g. sending prices, an emergency signal, or reporting energy use, there are existing mechanisms in OpenADR that implement the functionality. For a few capabilities, enumeration values specific to CTA-2045 have been added. A document that describes unambiguous mapping between the two standards is forthcoming.

End of Document