Battery Storage and Grid Integration Program

An initiative of The Australian National University

On the implementation and publishing of operating envelopes

evolve Project M5 Knowledge Sharing Report

Corresponding Author

Ben Weise
benjamin.weise@anu.edu.au

Battery Storage and Grid Integration Program
ANU College of Engineering and Computer Science
ANU College of Science

The Australian National University
Canberra ACT 2601 Australia
www.anu.edu.au

CRICOS Provider No. 00120C

Acknowledgement: *This Project received funding from ARENA as part of ARENA's Advancing Renewables Program and from the NSW Government.*

Table of Contents

1. Glossary

Term	Definition
ADMS	A term that has arisen recently and used in the industry to describe an Advanced DMS.
AIG	Australian Implementation Guide, an adaptation of CSIP in order to support Australian use cases.
Aggregator	An Aggregator is an organisation that provides an integration point and control mechanisms for a large number of DER assets
ANU	The Australian National University
BAU	Business as Usual.
CIM	Common Information Model, a standard for data exchange for network models, based on the IEC 61970, 61968 and 62325 family of standards.
Connection Point	The network location where a customer is electrically connected into the electricity system.
CSIP	Common Smart Inverter Profile, an IEEE 2030.5 implementation guide developed around the implementation of the standard in California.
DER	Distributed Energy Resource. Disruptive technologies being connected to distribution networks, including PV, EV, demand response solutions, storage and wind farms.
DERIAPITWG	DER Integration API Technical Working Group, an industry-led body formed to investigate and promote the standardisation of communications between DER and other parties.
DNSP	Distribution Network Service Provider. These are the organisations that own and operate electricity distribution network infrastructure.
DSO	Distribution System Operator; this term refers to the functions of Distribution Level coordination and optimisation of multiple DER aggregators in multiple markets, and connecting at the distribution network level.
End Device	An EndDevice represents either a 2030.5 client, or a device that is represented by a 2030.5 client.
HTTP	HyperText Transfer Protocol. The communications protocol used to connect to Web servers on the Internet or on a local network
NMI	National Metering Identifier (NMI) is a unique 10 or 11 digit number used to identify every electricity network connection point in Australia.
POST	A request method supported by HTTP, used to transfer information from a client to the server.
Utility Server	A utility server is generally run by the DNSP and provides an Application Programming Interface (API) that allows devices to send and retrieve data concerning their operation

2. Executive Summary

The orchestration of DER on the low-voltage network via dynamic operating envelopes represents a significant capability to increase network utilisation, but comes with additional data sources and calculations orders of magnitude beyond current DNSP practices.

The challenge of scaling to a solution that can manage (potentially) millions of smart controllable resources is significant. There is no single solution to this challenge – it requires careful consideration of architectures, protocols, and technologies used to implement such a system. This report considers, in particular, the interface between aggregators (or end devices) and DNSPs, and addresses the implementation of protocols, protocol extensions, and architectural decisions made in this implementation. Some of the many open questions involved in operating at scale are also discussed.

No single project will have answers to all of these questions, but the contributions from each project will build on previous experience to move the industry toward a secure, clean grid. To that end, we consider that the work of publishing open-source software to be vital, not just in this project, but within the industry more generally.

The experience of aggregators during this project has been very informative. Each aggregator has had a different experience of the project, with a number of factors contributing to this:

- The operating model of the aggregator, and how closely aligned it is with the goals of the project
- Capabilities under control of the aggregator, and the way in which that control is exercised
- The ability of the aggregator to align development work required for this project with other capability development

While aggregators generally view the development of standards in the industry as a net positive, there are significant barriers to the uptake of these standards. In the experience of this project, the development resources required to implement these standards is often a point of contention. For the industry to move forward, clear long-term signals are required to incentivize aggregators (and device manufacturers more generally) to engage in the process.

3. Introduction

This report has been prepared as part of the knowledge sharing requirements of the *evolve* project, under the ARENA agreement "2018/ARP154, Zeppelin Bend Evolve DER Project".
The primary objective of the evolve project is to develop and demonstrate a system for coordinating distributed energy resources (DERs) that will ensure the secure technical limits of the electricity distribution network are not breached.
More information on the evolve project can be found in Appendix B - Evolve project overview.
In meeting its primary objective, the Evolve project is also seeking to progress outcomes in the following three key areas:

1. Research into methods of calculating operating envelopes for DER assets. An operating envelope is a range of real and reactive power that controllable DER assets must remain within to avoid creating voltage or thermal overload issues in the electricity network. The underlying algorithms used for calculating operating envelopes will also support hosting capacity calculations;
2. The development and promotion of standards in relation to the modelling of electricity network asset and measurement data, and the exchange of these data models, and communications between DER assets and DNSPs; and
3. The development of open-source software building blocks that can be used by industry and academia to progress the development of new decision support and operational technologies for the integration of DER assets.

This report predominantly covers the development and promotion of standards in relation to the communication of registration, monitoring and envelope information between DER assets, aggregators and DNSPs.

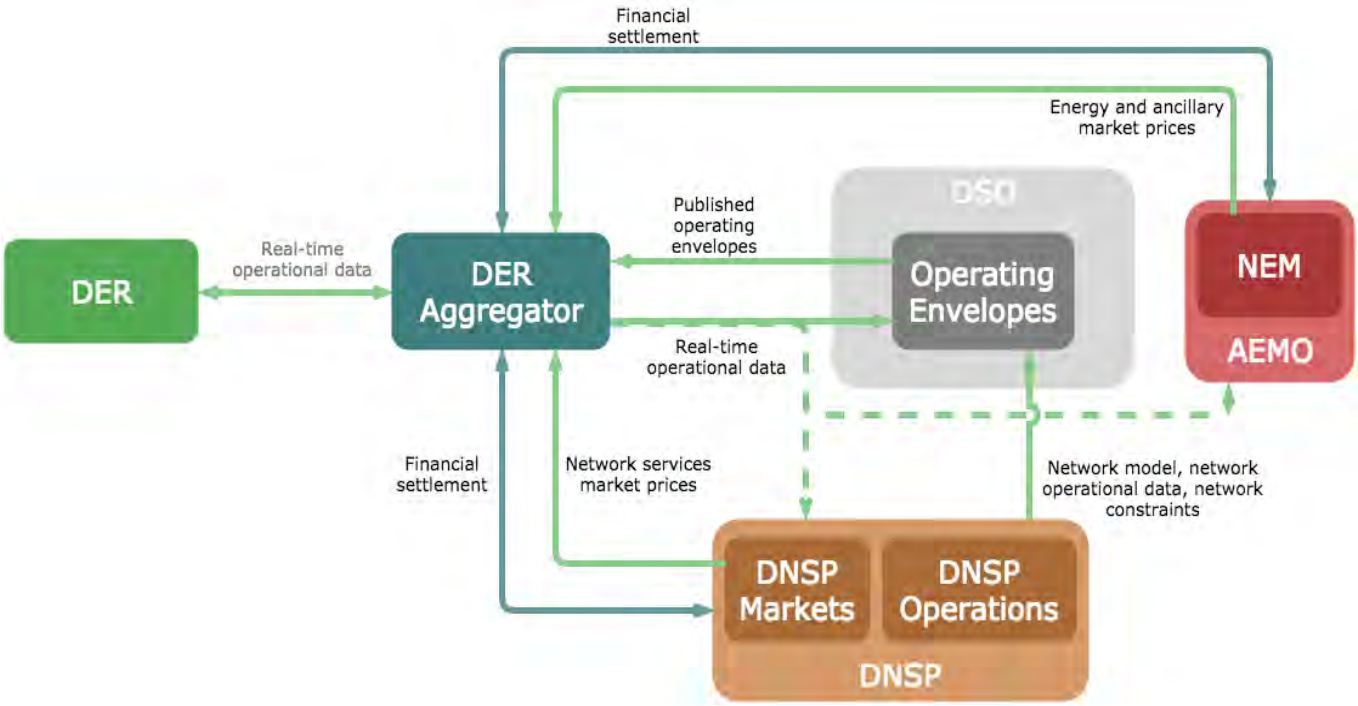


Figure 1 Overview of interaction between parties in the evolve system

The evolve project, in demonstrating the operational mechanisms by which envelopes can be used to co-ordinate the behaviour of distributed energy resources in the low voltage network, is building a platform on which DSO functionality can be developed and demonstrated. As such, the platform integrates with DNSP systems (such as ADMS and GIS systems), as well as 3rd party data providers and DER aggregators. This report is concerned with the integration between the platform and aggregators, both in terms of the technical implementation, and the overall experience of the aggregators involved in the project.

While the scope of the evolve project remains to demonstrate this capability between aggregators and DNSPs, consideration is also given to how these capabilities would interface directly with on-site DER assets. The report is organised into the following chapters:

Chapter 3	**The Envelope API**, covering details of the implementation of the aggregator API
Chapter 4	**Aggregator Integration**, detailing the experiences of the aggregators integrating into the evolve platform
Chapter 5	**Considerations for Scalability**, covering learnings from the project to date that can inform future work on scaling out this system to support larger numbers of smart DER

4. The Envelope API

Adoption of Standards in DER Communications

Aggregators have for some time offered access to device or virtual power plant management via proprietary APIs or interfaces. These have historically allowed DNSPs to access device data and in some cases procure network services from these aggregations of devices. This approach has limitations, since each DNSP must perform an integration against each aggregator to extract data and translate network support requests into each aggregator's own language. In this way, a DNSP becomes an 'aggregator of aggregators', as they endeavour to integrate every new aggregator operating on their network.

As the smart DER market matures, this approach limits the capacity of the DNSP to manage assets on its low voltage network, and develop and publish constraints that allow them to maximise network capacity utilisation. The alternative approach used in this project is to move towards standards-based communications between the DNSP and aggregators, with the DNSP-operated utility server being the basis for communication between parties[1]. While this approach increases the interoperability of devices within a DNSP, it is only effective if DNSPs use consistent protocols to achieve this integration.

This highlights the need for standardisation across these interfaces, in order to reduce the overall development effort required for aggregators to operate across multiple networks in, both in Australia and internationally.

Choice of Aggregator Communication Standard

While over the longer term, the growth and maturity of the controllable distributed asset industry may support the adoption of multiple standards in this space, it is vital that the electricity industry in Australia promotes interoperability between networks and projects, in order to give manufacturers, equipment vendors and operators, certainty in the development and operationalisation of capabilities to support dynamic connections.

In terms of capability, any envisioned system should support a scale of communication not previously seen in the energy sector, with visibility over these systems requiring granular reporting in a scalable platform. A number of features present in IEEE2030.5 that are not present in alternative protocols that make it suitable for this purpose and they include[2]:

- The ability to communicate default controls, to be enacted on loss of communications
- An object model that supports both aggregator-mediated (or passthrough) and direct-to-DER communications
- Monitoring functionality which provides both device visibility and network connectivity information
- The option to move to subscription-based communications and the potential for reduction in overall bandwidth requirements.

[1] This approach can also be naturally extended to direct-to-device communications; however this is not a goal for the evolve project.

[2] See, for example, this comparison of features of IEEE 2030.5 and OpenADR in relation to inverter management

Whilst previous work[3] has found this protocol to be generally suitable for these applications, it should be noted that no standard under consideration was found to be completely suitable for all use cases related to dynamic connection standards, both in the range of data structures provided for, and in the interpretation of those models. To that end, considerable work has been undertaken by the DER Integration API Technical Working Group in localising the standard for the Australian market. The evolve project will continue to lead and contribute to this working group, with an implementation guide targeted for release in March 2021, and work on the expansion of this guide to incorporate more industry use cases, as well as improvement in the testing procedures to measure conformance with the guide.

IEEE 2030.5 API Implementation

The evolve project took a staged approach to the implementation of the IEEE 2030.5 API. Noting that the standard's adoption and use in this space (particularly in Australia) is very recent and evolving, we initially developed an API based on the data structures used in the standard. In parallel, the implementation of a conformant utility server was begun. This version of the API is in the process of being open-sourced (and will be made available prior to the end of the project).
There is considerable interest in the adoption of this standard across the Australian industry, with most networks either exploring or implementing an equivalent utility server. Over the course of this project, and through discussions with other DNSPs, the structure of this implementation is continuing to evolve, including:

- Separation into core, administration, and extension functionality
- Development of a plugin system that allows data to be more easily integrated with external systems
- Separation of envelope publishing functionality to make the system more modular and more applicable to a broader range of use cases

System Components

The core of the API is built using the FastAPI Python framework. Python is a general-purpose programming language with wide software-industry application and a vibrant open-source community following. It is consistently rated in the top three programming languages (according to Github[4], TIOBE, and Redmonk[5]), and was rated as the top programming language in 2020 by IEEE[6].
FastAPI is a "*modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints".* It was chosen for the implementation due to its overall speed (both in terms of performance and in developer speed), while incorporating robust data validation and standards-based documentation.

[3] California Rule 21 selection of IEEE 2030.5 as the default communications protocol, demonstrated its overall utility in this scenario. The DER Integration API Technical Working Group ultimately chose the protocol as the basis for communications in its implementation guide in early 2020.

[4] Github statistics, 2021

[5] Redmonk language popularity, 2020

[6] IEEE programming languages, 2020

Data persistence is provided through interfacing with Postgresql[7]. Documentation is generated in OpenAPI[8] format, a common standard used in API development.
Testing is conducted on multiple levels, from unit testing in Python using pytests to integration testing using Postman and Newman[9] integrated with Github Actions[10].
Authentication and TLS termination is performed at the load balancer / reverse proxy, based on self-signed certificates. This allows for the security implementation to be decoupled from the utility server, enabling flexibility in deployment. The evolve project uses *nginx*[11] *Open Source* for production system load balancing and authorisation based on IEEE 2030.5 requirements.

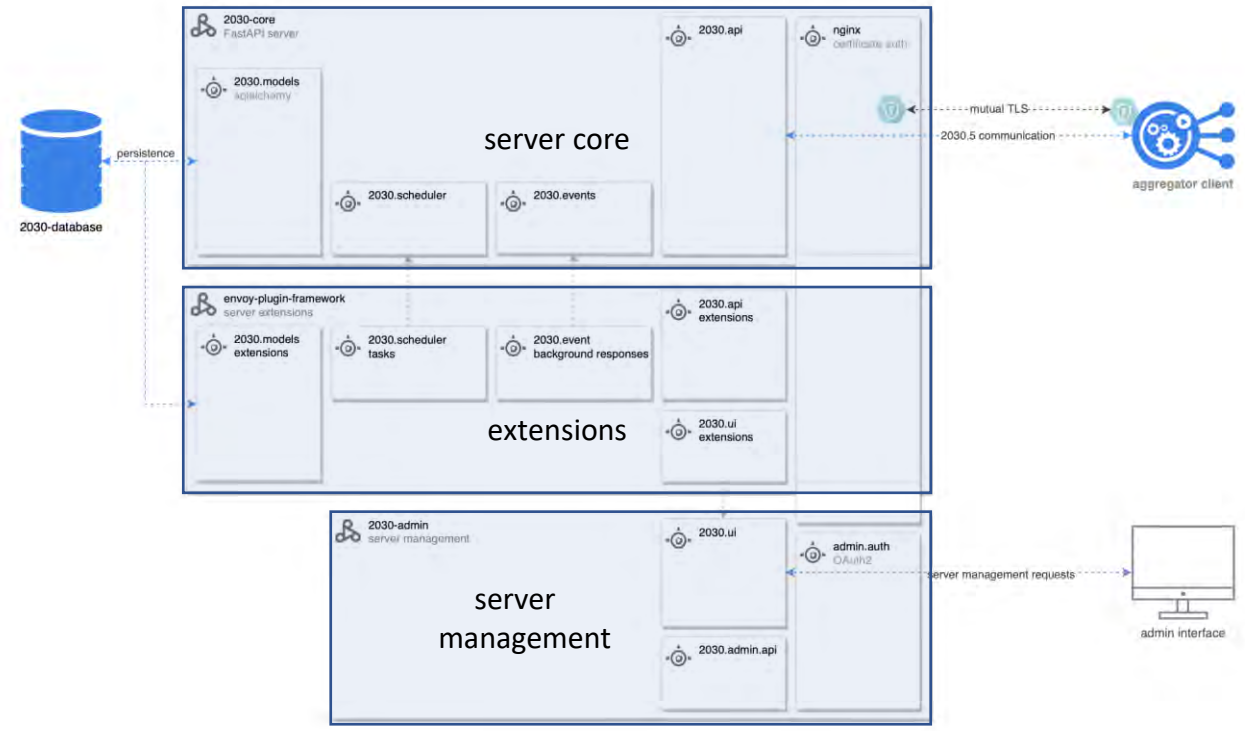


Figure 2 2030.5 server components.

Interface Extensions

[7] Postgresql is an open-source object-relational database system that has been in active development for over 30 years.

[8] OpenAPI was previously known as Swagger and is an industry-standard format for specifying application programming interfaces.

[9] Postman is a widely-used tool for interacting with web APIs. Newman is a command-line tool that allows for testing based on collections of requests built in Postman.

[10] Github Actions is a software build, test and deployment tool integrated with the code hosting platform Github.

[11] Nginx Open Source is the open source load balancer built by nginx. Commercial extensions are available but not required.

While IEEE 2030.5 serves as the basis for the API implementation, not all required functions are supported by core standard. A key concern of the standard is that it is focussed on device behaviour, whereas Australian DNSPs are more concerned with managing overall site behaviour. A key principle of the evolve project is that aggregators should be free to control systems behind the meter in whatever way they consider optimal, providing that the overall site limits are not breached.

As a result of these differences, a number of extensions have been made to the interface. These changes have been designed to be compliant with the standard[12], although further work may be required to demonstrate conformance. These extensions are in the process of being incorporated into the Australian Implementation Guide.

In-band Device Registration

CSIP specifies that initial device registration is an out-of-band procedure that, for an aggregator, involves the provision of information to the DNSP in an unspecified manner. In the longer term, the DER register may provide sufficient information to perform this step. For the evolve project, we consider that the initial registration can be performed through the 2030.5 protocol, whereby the aggregator POSTs information to the EndDevice endpoint in order to register the device under its control. From the Australian Implementation Guide:

> As in CSIP 6.1.5, an aggregator is a special EndDevice to the utility server. It is assumed that these devices will be registered through an out-of-band process between the aggregator and the utility server.
>
> An aggregator will be differentiated from other *EndDevices* by the *DeviceCategory* field having bit position 18 set (virtual or mixed DER). DER devices under aggregator management will be represented as separate *EndDevices*. When an aggregator queries the *EndDeviceList* endpoint, it will receive a list of at least 1 *EndDevice*. The first device in the list will correspond to the aggregator *EndDevice* (*SelfDevice*). The aggregator *SelfDevice* does not function as a DER resource and has no linked DER devices. As it does not represent any controllable resources, the aggregator *EndDevice* will have a *DeviceCategory* of 0x0000 (no *DeviceCategoryType* bits set).
>
> In the case of aggregator-mediated communications, where the resource communication is proxied through the aggregator and the aggregator acts as the 2030.5 client, the aggregator may act to register these devices directly with the server, without prior information being separately communicated. The aggregator *EndDevice* must still be registered with the server in an out-of-band procedure.

Site Registration

In order to appropriately assign devices to topological groups, the utility server operator must know the location of the device relative to the network. While the standard allows for the communication of the device's GPS location, this may be insufficient (or insufficiently accurate) to locate the device on the network. Particularly for residential locations, an aggregator may have access to the site's NMI, which is a more reliable means of identifying the site.

The 2030.5 standard (and CSIP implementation) specifies that additional information be transmitted out-of-band in order to locate the device on the network. However, this introduces additional overhead, requiring manual data collation and input. This extension is an optional method for the communication of this information that allows for the process to be completed entirely through the utility server. It involves the

[12] See chapter 11, IEEE Std 2030.5-2018

provision of an additional ConnectionPoint object, with an associated API endpoint and payload structure. An aggregator (or, if enabled for direct communication, the device itself) may POST to the EndDevice ConnectionPoint endpoint the following payload:

```
<ConnectionPoint xmlns="urn:ieee:std:2030.5:ns">
  <evolve:meterID>1234567890</evolve:meterID>
  <evolve:connectionPointID></evolve:connectionPointID>
</ConnectionPoint>
```

The *meterID* value will, in the Australian context, generally be the National Metering Identifier[5]. In the case where use of this value is not permitted, the utility server operator may provide an alternative identifier (here identified as the *connectionPointID*)

Envelope Specification

IEEE 2030.5 is predominantly designed around DER settings management and control. While the concept of DERControl and DefaultDERControl in the data model align closely with the temporal aspects of operating envelopes, no attributes of DERControl exist that accurately convey the power limits of an operating envelope. For the simplest real power envelope[13], two additional attributes are required: opModMaxImportW and opModMaxExportW, to denote the real power (site) import and export limits, respectively.
The specification of a DERControl signal then becomes:

```
<DERControlList all="2" href="/derp/0/derc" results="1" subscribable="1" xmlns="urn:ieee:std:2030.5:ns">
    <DERControl>
        <mRID>02BE7A7E57</mRID>
        <description>Example DERControl 1</description>
        <creationTime>1341446390</creationTime>
        <EventStatus>
            <currentStatus>1</currentStatus>
            <dateTime>1341532800</dateTime>
            <potentiallySuperseded>false</potentiallySuperseded>
        </EventStatus>
        <interval>
            <duration>86400</duration>
            <start>1341446400</start>
        </interval>
        <DERControlBase>
            <opModMaxImportW>10000</opModMaxImportW>
            <opModMaxExportW>-5000</opModMaxExportW>
        </DERControlBase>
    </DERControl>
</DERControlList>
```

More complex envelopes involving reactive power compensation may be implemented as additional DERCurve attributes. Clarification is being sought from the IEEE 2030.5 working group as to the most appropriate implementation of more complex envelopes.

[13] See ARENA Report – On the Calculation and Use of Dynamic Operating Envelopes, ANU 2020

5. Aggregator Integration

The aggregators partnering in the evolve project represent a significant cross-section of the aggregator market, with different business models, hardware, software and communications protocols. As such, the evolve project provides an opportunity to examine in detail the experience of aggregators integrating against the dynamic operating envelope platform.

Understanding Barriers to Standards Adoption

While future interoperability is a key consideration for the evolve project, and the use of standards is a key driver of interoperability in the Australian electricity industry (as evidenced by this project's use and development of standards and profiles that may be used in future work), there are considerable barriers to the adoption of standards in this space. Some of these, encountered by the evolve project, include:

- Increased resourcing required to move from proprietary approaches to adoption of standards
- Contention for resourcing within smaller aggregators
- Fit-for-purpose of existing standards, particularly for newer applications such as dynamic operating envelopes
- Relative immaturity of the smart distributed energy resource market

While the project aims to push forward the introduction of standards in the communication mechanisms and data structures used in the DER space, we have taken a pragmatic approach to the introduction of conformant systems. For the aggregator integration API, this has involved the development of 2 utility servers:

- An initial implementation server which presents common IEEE 2030.5 data structures in a more accessible way
- A second, conformant server that incorporates the core envelope functionality in a conformant server.

Given that the global adoption of standards in this space is relatively immature, this was a decision to reduce the risk and overhead in first establishing the utility of a consistent API for this communication, before moving towards full conformance with existing standards.

Aggregator Capabilities

The aggregators represented in the evolve project offer overlapping capabilities for control and management of smart DER. However, the mechanisms by which these capabilities are controlled vary significantly, and this poses some challenges for the integration of each aggregator in the project. Some key differences between aggregators include:

Location of Control System

Aggregators differ in where the control system is physically located – with some aggregators running the control system in the cloud (with command being communicated to the DER over the internet), while others physically locate the control system hardware on the customer premises. Even for those with on premise hardware, some aggregators control systems are tightly integrated with the controllable resource, while others may control devices through a (partially) standardised interface.

Integration of Control System

While some aggregators tightly couple the control system with supported hardware, others may support a range of systems through an abstracted interface. While this choice has implications for the configuration of hardware, it also has implications for the control capabilities that are supported. For example, one aggregator which supports the control of multiple inverter manufacturers, may be limited in terms of support capabilities based on the particular inverter model and firmware. Further complicating the exposure of device capabilities, particular functionality may only be available under certain electrical configurations (for example, reactive power control may not be available where multiple inverters are installed).

This presents challenges in managing device capabilities and the provision of envelopes to devices based on these capabilities – this represents a leaky abstraction, whereby the utility server may need to know more about the on-site configuration than would otherwise be necessary.
In addition, some control capabilities may be available by the controllable device itself (i.e. through firmware capabilities), while others may be available through aggregator control systems. For example, many inverters support export power limits based on site output natively, so support for this by aggregators as part of an operating envelope specification would in general be less complex. However, import power limits are not generally supported in the same manner, so developing this capability in the outer control loop[14] would be more complex.

Types of Controllable System Capabilities

Given the range of supported system configurations, it is important to understand the variety of control capabilities available to any system implementing dynamic operating envelopes. Behaviours that can support the implementation of envelopes include:

- **Solar curtailment:** solar output can be limited to ensure that the connection point export limit is not breached (noting that this will have a financial impact on the customer)
- **Battery charge/discharge scheduling:** battery charge behaviour can be modified such that the operating envelope is not breached, with minimal financial impact on the customer[15]
- **Reactive power control:** reactive power voltage compensation arrives naturally from the solution to optimal envelope allocation in a voltage-constrained network. Thus, systems that are able to adjust reactive power can compensate for voltage rises due to generation by acting as a reactive

[14] In general, the inner control consists of inverter-level firmware control, and is preferable to monitoring and control by external systems due to the speed of monitoring. The outer control loop is calculated by the aggregator's hardware or software and usually updates at a slower poll rate.

[15] Note that, in the context of forecast error, there may be some financial risk to the customer as a result of this re-scheduling – for example, unscheduled loads affecting the overall forecast. Additionally, this requires a 24-hour envelope schedule in order for the aggregator to determine the most appropriate strategy.

power sink (provided that thermal limits are not exceeded). Most reactive power compensation is at minimal cost to the customer.

Note that this set of capabilities is not exhaustive to the general case, but is illustrative of the capabilities provided by systems in the evolve project. Within the project, not all systems support all capabilities. Table 1 shows examples of unsupported capabilities in different systems, and the cause thereof.

Unsupported Capability	Reason
Solar Curtailment	Solar inverter not being under aggregator control
	Lack of inverter firmware support
	Lack of integration with supporting inverter
Reactive Power Control	Not exposed in firmware
	Electrical configuration

Table 1. Examples of unsupported capabilities and reasons for lack of support

As support for dynamically controlled devices operated behind the meter grows, operating envelope generation must be able to support new types of controllable devices. While this is partially catered for through the representation of device capabilities in the aggregator interface, future device capability support may require extensions to the standard.

Aggregator Operating Models

The use of the DNSP-operated utility server as the source of dynamic operating envelopes involves, for some aggregators, a significant shift in the operation of their systems. Historically, as the market for virtual power plant (VPP) services grew, aggregators were able to differentiate their products by providing additional network services based on the aggregation of capabilities represented by a virtual power plant (VPP). Commonly, a network would make requests to a VPP provider based on groupings of resources that were managed by the aggregator on behalf of the network.

The use of a utility server transforms this method of operation. While a DNSP may be able to procure services from an aggregator in this fashion, where multiple aggregators operate in an area, the DNSP would be required to integrate against multiple aggregators using proprietary interfaces (see Figure 3 for a depiction of the change in architecture between current operation and the utility server model, and Figure 4 describing the inversion of control and communication initiation this entails).

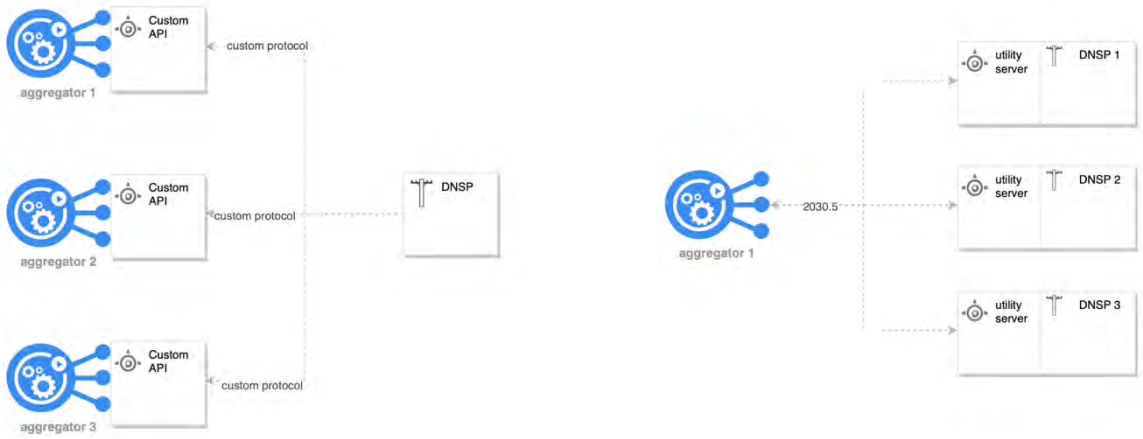


Figure 3 Left: where aggregators expose an API for integration, each DNSP integrates against each aggregator's custom protocol. Right: under the utility server model, each aggregator integrates against each DNSP it operates within. Communication is standardised between networks, which limits the number of integrations needed to be developed by the aggregator.

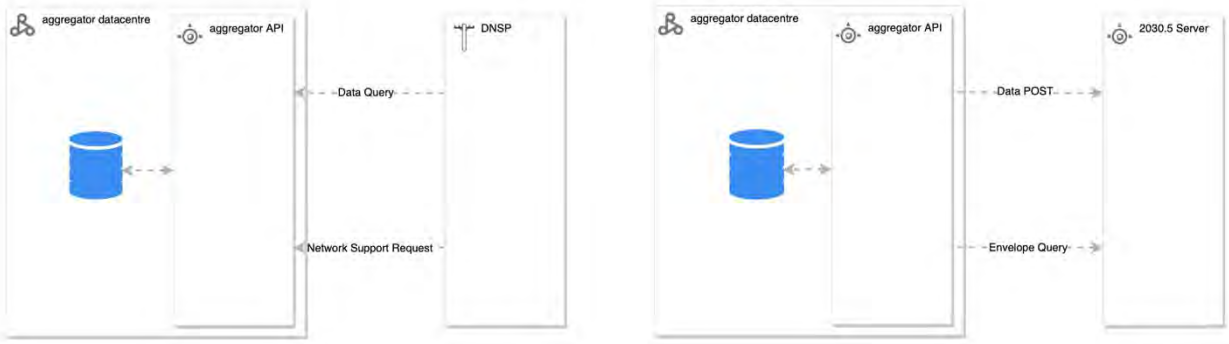


Figure 4 Left: in the current architecture (aggregator-developed API), the DNSP initialises any data queries or network support requests. Data is managed by the aggregator. Right: under the utility server model, the aggregator initialises queries, by POSTing data to the utility server and retrieving envelopes.

For some aggregators, this represents a significant change to the model that they operate under. Figure 5 shows an aggregator model where the aggregator manages device groupings, and exposes only virtual power plants to provide network support services. The shift to externally-managed groupings and allocations, although conceptually similar, presents a significant shift in the operating model of the aggregator. The main options for implementation were:

- Develop the capability to sync aggregator-managed VPPs to an external source (the utility server), and cede control of allocation within the VPP to the server.
- Create a virtual power plant for each trial system, and pass system envelopes via these individual VPPs.

The first option was further complicated due to the fact that the initial API implementation does not expose device groupings, and so the second option was deemed more appropriate for the trial.

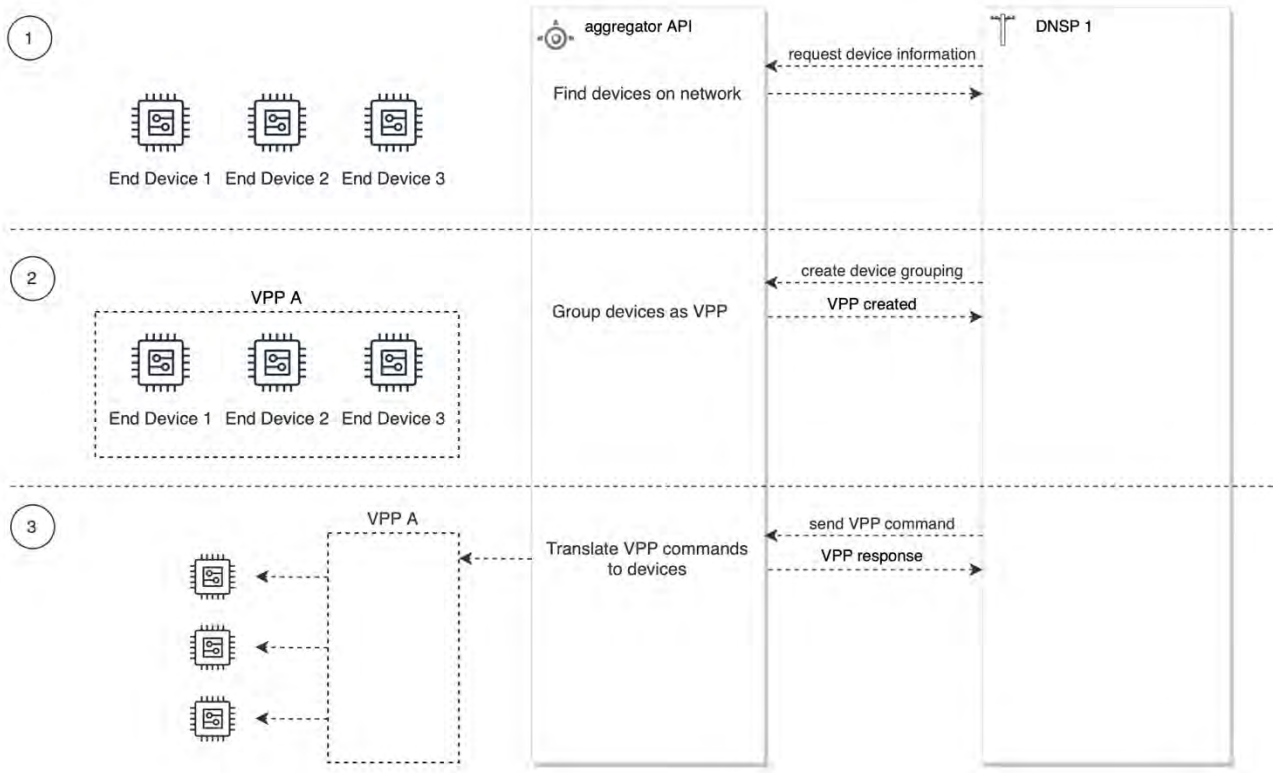


Figure 5 In this operating model, the aggregator manages device grouping and exposes only VPP actions to the DNSP, by:
1. Allowing discovery of relevant devices
2. Managing the aggregation of those devices as VPPs
3. Exposing interfaces to allow for the command of VPPs

Alignment of Capability Development with Aggregators

In the experience of the evolve project, there have been examples where the development of these capabilities has aligned well with other capabilities being deployed by aggregators, as well as instances where the integration with existing capabilities was significantly more complex (for example, for aggregators with a strong focus on VPP interactions by networks, with VPPs managed by the aggregator).

One aggregator was able to include operating envelope development within a broader body of work, aligning the data structures required to manage and communicate operating envelopes with other capabilities. As a result, they have been able to release a scheduling API for their systems that incorporates similar data structures, which has enabled them to expose greater functionality to their customers while also meeting their requirements for the project.

From these experiences, we believe that providing long-term certainty around the deployment of operating envelopes in Australian networks will allow aggregators (and device manufacturers) to better align their development and operations with the capabilities required to operate with Australian DNSPs.

Open-Source Contributions and Community

An important contribution that this project makes to the overall interoperability picture in the electricity industry is through the contribution of open-source software. To date, there has been limited access to open-source software and tools that assist with the development and integration of standards-conformant communications mechanisms. The evolve project is in the process of open-sourcing the following

components that will assist in the implementation and uptake of standards-based communication between aggregators and utilities:

- An IEEE 2030.5 utility server core[16]
- Tools for the simulation and validation of server/client behaviour[17]
- An administration interface for the management of out-of-band processes (including device registration)

The fostering of an open-source community is a vital component to creating a useful open-source software system. Through the evolve project, and discussions with other networks and organisations, we recognise that the use cases for this software go beyond the scope of the evolve project. We have found that, particularly for the IEEE 2030.5 standard, the lack of open-source components, as well as documentation and implementation guidance, to be a severe impediment to the development of conformant systems. This is, particularly for smaller aggregators or system builders, a large barrier to integration, and we believe that our contributions in this area will be a significant contribution to the community.

In our implementation (and as discussed in IEEE 2030.5 Technical Considerations), we have found that, even for demonstrably useful technologies, without broader support (including open-source implementations) and a strong community, uptake may be limited.

6. Considerations for Scalability

Deployment of distributed energy resources has seen strong growth in Australia (insert stats here). However, the uptake of smart, controllable DER has to date been limited due to technical, economic and social barriers . Therefore, any solution to the management and co-ordination of DER needs to account for a large increase in smart DER, and the associated increase in communication and processing that this entails.

Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands[18].

The overall scalability of a solution is determined by the ability of all components of the system to adapt to meet increased demand – a high-performance web server may be constrained by an inability to write data to persistent storage at the same rate. It is often difficult to identify bottlenecks without rigorous integration testing, and bottlenecks may be particularly sensitive to hardware or configuration changes. In this section, we discuss some of the considerations made in the evolve project to scalability.

Envelope Design

While the approach to envelope design in the evolve project is to take into account multiple factors, including network topology and operating conditions, DER location and capabilities, as well as principles of equity in

[16] Github repository

[17] To be made public in Q2 2021

[18] Gartner, 2021

allocation, additional factors can arise from the way in which these envelopes are stored and communicated that affect system scalability.

Under IEEE 2030.5, *End Devices* are grouped logically into *Function Set Assignments*, to which operating envelopes are assigned. It is anticipated (and indeed, recommended in the CSIP) that these assignments be made according to network topology. From this, it naturally arises that allocating envelopes equally among devices in the lowest level of the topology (generally beneath the distribution transformer) reduces the amount of communication overhead between the DNSP and aggregator when compared to allocating envelopes individually to each system, since the same envelope would be assigned to multiple systems. (Note that individually calculated envelopes also requires additional groupings beyond those specified in CSIP, where each device essentially belongs to its own group). Allocating envelopes at a higher level in the topology would also reduce overhead; however a trade-off must be made between the efficiency of the allocation of network capacity and the size of each grouping.

Note: In the case where communication is direct to each End Device, there is no commensurate reduction in communications, since each device is responsible for retrieving its own groupings and envelopes

Approaches to Envelope Aggregation

We make the distinction between 'grouped' envelopes (where the platform assigns a defined envelope to each device, where devices are usually grouped topologically), and 'aggregated' envelopes, where a set of devices is given an overall envelope, and the allocation is managed by the device aggregator.

To date, the evolve project has been concerned with direct calculation of envelopes for each device. Under this approach, while neighbouring devices may receive an equal allocation, there is no ability for an aggregator to manage, or 'transfer', allocation of capacity between systems in order to optimise their behaviour. As previously mentioned, this approach aligns with the manner in which envelopes are distributed under IEEE 2030.5.

Aggregators have expressed interest in being able to manage the behaviour of their systems in aggregate, in order to optimise individual behaviour against network constraints with more flexibility. This approach has a number of potential complexities, including:

- Envelopes are not strictly additive, as different devices may have different voltage impacts on the network. Thus, there is tension between allowing an aggregator greater flexibility (aggregating to a larger set of devices), and maintaining network constraints (aggregating to smaller sets of neighbouring devices)
- Due to the non-linearity of voltage impacts, transmitting information about the way that an allocation method affects the overall size of the envelope adds additional complexity to the required aggregator allocations
- Such a scheme is likely at odds with future network considerations for dynamic customer connection agreements
- While the concept of aggregations is supported at the protocol level (in that aggregations of DER can also be registered through 2030.5), this reduces the visibility of device and site behaviour that is required for dynamic envelope calculations

As such, the project has limited calculations to envelopes directly assigned to sites and devices.

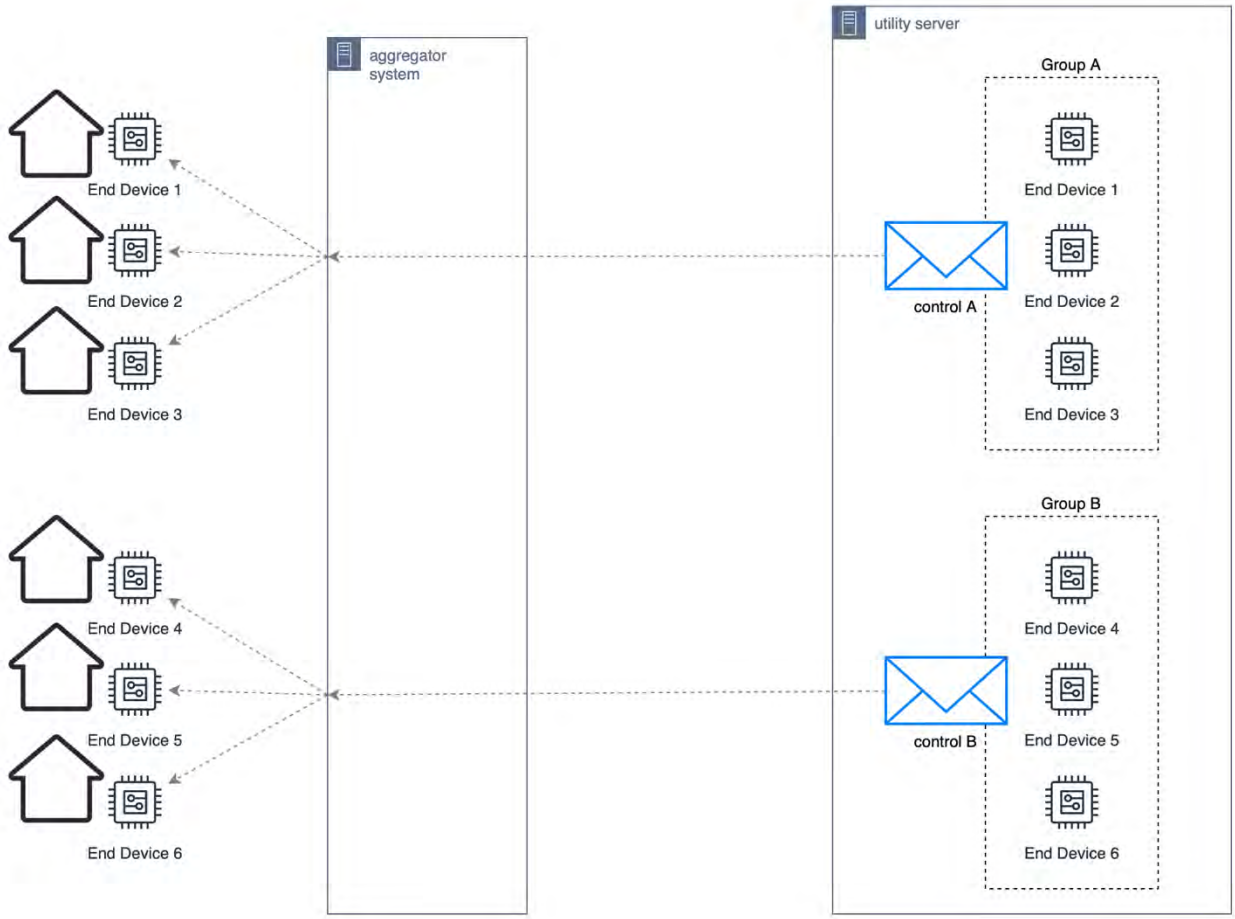


Figure 6 ***For equal (grouped) allocation, an aggregator retrieves each envelope specification once, before distributing to end devices.***

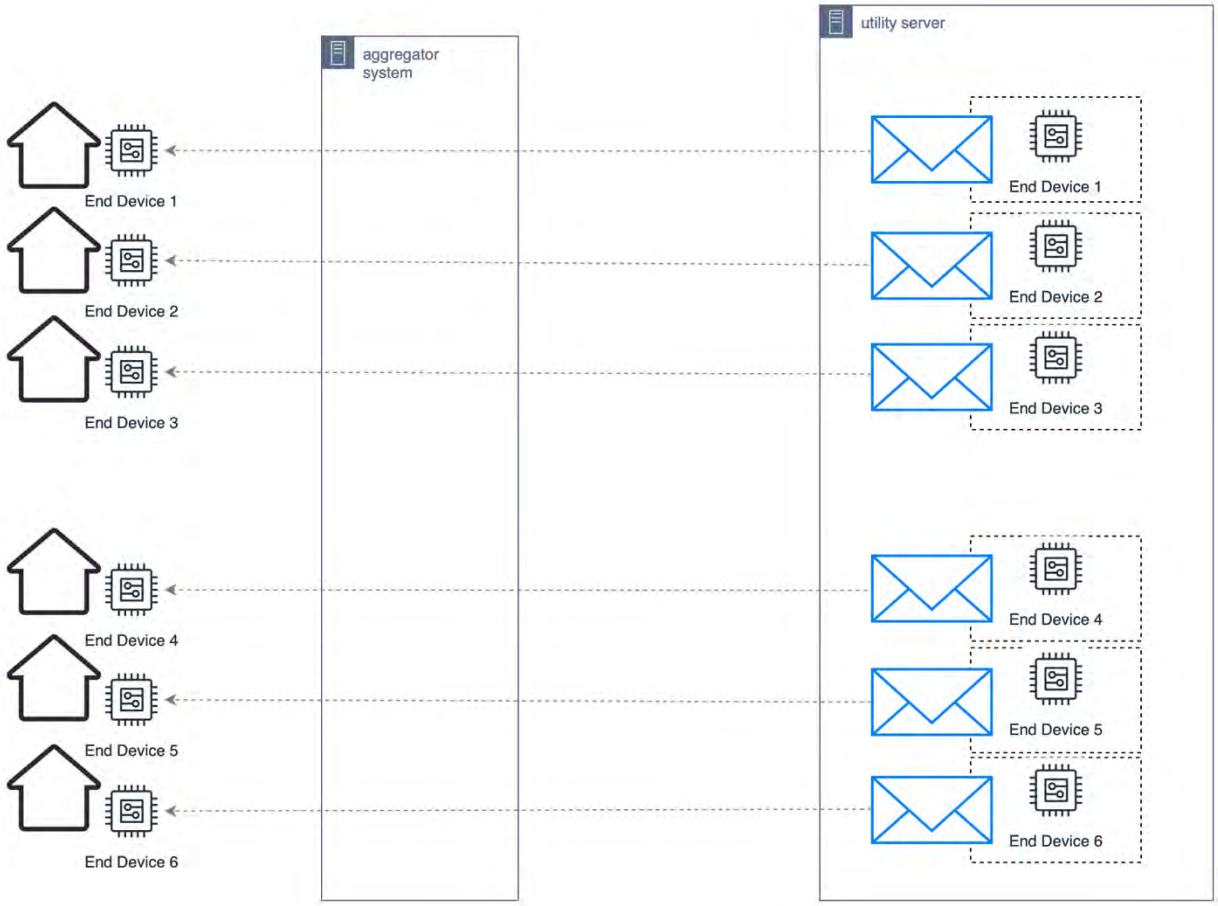


Figure 7 ***For unequal (individual) allocation, the aggregator retrieves an envelope for every device.***

Data Requirements

The evolve project requires systems to report monitoring information at 5-minute intervals, and this is reflected in the data requirements of the Australian Implementation Guide. The communication of this information serves two purposes:

1. The monitoring data provided by these sensors contributes to the system's understanding of network conditions, feeding into the operating envelope calculations.
2. The presence of monitoring information indicates that a system is online and able to respond to any DERControls passed through to it.

As a corollary to the second point, the AIG also describes the use of this POSTing mechanism as a means for the client to check connectivity to the utility server, with the client enacting default controls whenever communications are interrupted.

As the number of smart DER systems communicating with a utility server grows, the amount of data sent and received will quickly grow. For example, 1 million devices reporting back voltage, power and reactive power (minimum, mean and maximum) for both the site and device would require the processing of 60,000 data points per second.

While monitoring data contributes significantly to the overall bandwidth usage of a large-scale DER co-ordination system, the dual purpose of regular monitoring data updates (both for the data itself and as an indication of network connectivity status) mean that it is a necessary part of most dynamic capacity

calculations, and thus any means of reducing the resources required. The most significant means of reducing bandwidth requirements for monitoring data from aggregators is to introduce bulk monitoring API endpoints, to reduce the number of requests required. Note that this only applies to aggregator-mediated communications; a bulk endpoint will not improve communications overhead for direct-to-device communications.

Handling API Load Heterogeneity

Given the requirement for data to be collected at (market-aligned) 5-minute intervals, this introduces a high degree of correlation in requests. Additionally, some aggregators have an architecture that deals more efficiently with batch processing, so all data coming from a single aggregator closely time-aligned. There are a number of potential methods to handle these load spikes:

- Over-provision hardware to deal with these large spikes (noting that this hardware may be under-utilised the majority of the time)
- Implement appropriate error handling on the server to inform the client that it is not currently able to handle the request
- Use scalable resources to match server demand with appropriate resources

The evolve project deploys the API on cloud resources in order to take advantage of scalable infrastructure, while minimising the costs associated with under-utilisation of hardware. This approach minimises the need to handle errors associated with server load. While this is an effective approach for the project, it should be noted that the IEEE 2030.5 provides guidance on the implementation of resource constraint errors on the server:

> *A file server (FS) SHOULD maintain internal estimates of its current resource usage. If an FS determines that it is unable to service an incoming HTTP request, the FS SHOULD issue a response indicating 503 error and SHOULD include the Retry-After entity header (providing guidance to the loading device (LD) for retry attempt). The Retry-After entity header SHOULD provide a best estimate as to when the FS expects it will be able to service the LD request.*
>
> *An LD SHOULD implement handling for 503 errors and SHOULD implement handling of the Retry-After entity header. If the LD does not support processing of the Retry-After entity header, the LD MUST wait at least 30 seconds before retrying the file content request.*

Communications Outages

As mentioned above, the communication of interval data also serves to inform the utility server about the communications status of devices. It is assumed that any device that does not report monitoring data is `offline`, and thus unable to respond to any updated envelopes. In any large distributed system, there will be considerable numbers of devices offline at any point in time, particularly considering the reliance on consumer internet[19] for much of the communication between devices and DNSPs. In the calculation of envelopes, care must be given to the way in which envelope allocation handles these devices, and

[19] While some devices may communicate over mobile internet or mesh networks, it is anticipated that the additional cost involved will mean that the majority of devices connect to the household's consumer internet.

assumptions made about their behaviour during an outage. In the AIG, default behaviour for a device is described; however given the scope of the project, this aspect of device behaviour is not being trialled.

IEEE 2030.5 Technical Considerations

The choice of standard for communication protocol has a significant impact on the overall system architecture and scalability (in terms of compute, bandwidth and storage considerations). IEEE 2030.5, being based on HTTP, is a standard client/server configuration that uses robust, ubiquitous internet communication technologies. While this precludes the use of newer technologies that may provide performance benefits (such as websockets), the underlying technology has been battle-tested and demonstrated at web scale.
Specific design choices within the standard also have impacts on scalability. In this section we discuss these as they relate to the evolve project, and dynamic operating envelopes more generally.

Data-Interchange Formats

IEEE 2030.5 uses XML (eXtensible Markup Language) as the basis for its data interchange format. The use of XML as the underlying data serialisation mechanism also impacts architectural decisions and choice of technology. While historically, XML has seen broad use in enterprise platforms, it is rarely the first choice for web technologies, and experience with XML in the sector is relatively limited. XML is generally recognised as a verbose, information-sparse data-interchange format. Transmitting raw XML is therefore quite inefficient, in terms of bandwidth required for the amount of information transmitted – an encoding or compression scheme is required. The IEEE 2030.5 standard specifies EXI (Efficient XML Interchange) as the encoding scheme used to both limit bandwidth and processing power required for decoding. However, other schemes exist - for example, for the transmission of similarly information-sparse data such as HTML over HTTP, gzip compression is commonly used.

EXI (Efficient XML Interchange) is a is a binary XML format for exchange of data on a computer network developed by the World Wide Web Consortium's Efficient Extensible Interchange Working Group, and is designed to reduce computational overhead and bandwidth in the processing and communication of XML data structures.

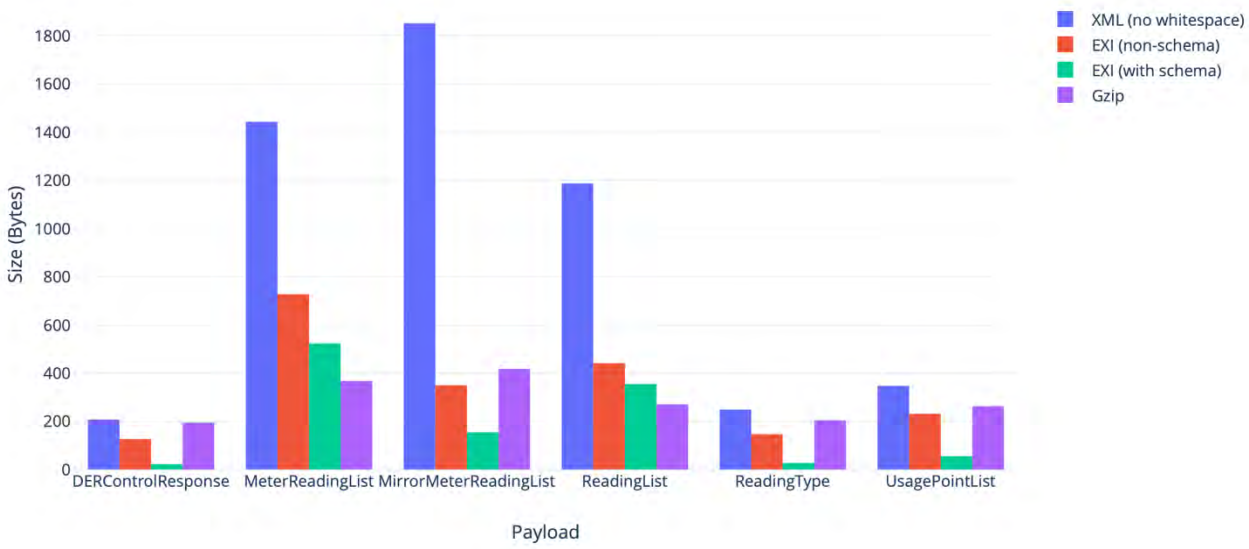


Figure 8 Comparison of payload size with EXI encoding and gzip. For most payloads, EXI (with schema) out-performed gzip.

Figure 8 shows a comparison of XML payload sizes using EXI (both with and without schema) and gzip (the standard compression used on most internet traffic). Example payloads are taken from section the Annex C examples in the standard. Notably, EXI tended to perform significantly better on smaller payloads, while on larger payloads the difference was not as clear – in some cases, gzip out-performed EXI (due to the lack of compression specified in the IEEE 2030.5 standard).

Notably, a common request from aggregators in the evolve project was for 'bulk' data endpoints, where data from multiple devices could be sent in one request. This is not available in IEEE 2030.5. Considering this requirement, we also looked at the payload size under the following scenarios:

Representative data for a 5-minute period for a single device: in this scenario (consisting of minimum, mean and maximum power, reactive power and voltage data), gzip performed similar (within 10%) of EXI.

A simulated 'bulk' endpoint with 5-minute data for 10 devices: in this simulated payload (consisting of minimum, mean and maximum power, reactive power and voltage data for 10 devices), gzip compression out-performed EXI, being half the size of the equivalent EXI payload.

This indicates that, for the common scenarios for aggregators sending monitoring data to the utility server, the commonly used gzip compression is a relatively effective means of reducing bandwidth.

The 2030.5 standard requires that EXI 'shall be supported' by the server and either XML or EXI supported by the client. Given the lack of broad support for EXI for a range of programming languages[20], the requirement for support of EXI may be excessively burdensome and limit the development of 2030.5 systems. Given that there exist much more widely supported means of reducing bandwidth usage and communications overhead, it is advisable to make this a non-mandatory option for any implementation.

Subscription/Notification Model

While the IEEE 2030.5 architecture is a client/server architecture (with the end device initiating communication), it also presents an (optional) subscription/notification model. Under this model, a client subscribes to a resource and receives notifications when this resource is updated. This model has the potential to reduce communication overhead; however, there are a number of factors to consider:

- Subscription/notification must be supported by both the server and the client
- For the client to receive notifications, they must be running their own service that can be reached by the utility server. This is particularly problematic in consumer internet environments, since home routers will often limit the ability to expose services externally (due to either firewall rules or Network Address Translation issues)
- This setup requires additional authentication work, since it operates outside of the authenticated mTLS connection
- While it may be useful to subscribe to some resources, linked resources will still need to be retrieved through the regular mechanism as they won't be contained in the notification body
- Regular polling is still required to ensure, for example, that no new alterations were made to device groupings

[20] Currently, implementations exist in C/C++, Java, and JavaScript (although the JavaScript implementation is incomplete). As an indication of popularity, StackOverflow (a public question and answer knowledge base for developers) has 22 questions tagged `EXI`. In comparison, there are over 200,000 questions tagged with `XML`.

The potential benefit is greatest for bandwidth constrained end devices, however these scenarios most commonly present technical difficulties in implementation. For aggregator-mediated devices, appropriate group management is a more tractable path to reducing communications bandwidth, particularly given the requirements around the provision of monitoring data to the utility server already represent a significant communication overhead. As such, while this mechanism may contribute to system scalability, it should not be relied upon in the general case.

Security Considerations

IEEE 2030.5 uses mutual Transport Layer Security[21] (mTLS) authentication to ensure communication occurs only between authenticated parties. The TLS negotiation of a secure channel necessarily involves additional overhead

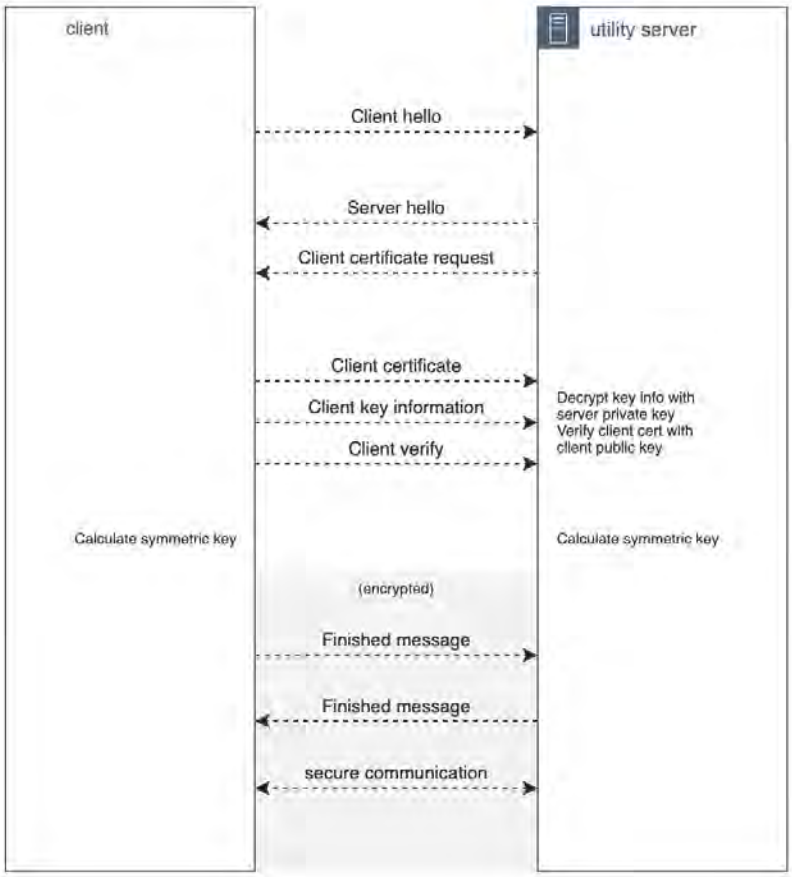


Figure 9 Steps involved in creating a secure mTLS communications channel.

The overhead introduced by TLS negotiation can be somewhat minimised via the implementation of session resumption techniques.

There are two mechanisms for session resumption using TLS 1.2[22] – either using session identifiers or session tickets. Under session ID resumption[23], the server must keep track of previously negotiated sessions using a

[21] https://en.wikipedia.org/wiki/Mutual_authentication#mTLS

[22] Note that TLS 1.3 (RFC8466) uses a separate session resumption mechanism, and does not support the two described methods. However, TLS 1.2 is required according to the IEEE 2030.5 standard.

[23] RFC5246

session ID, which allows it to resume encrypted communication with a client based on a previous session. Under this scheme, both the server and the client must track the session ID in order to support session resumption. For horizontal scalability, this adds complexity to the server architecture, as any load balancers must support a shared memory model to allow for the sharing of session identifiers.

Using session tickets allows the server to statelessly resume a previous connection. This, however, comes at the expense of perfect forward secrecy[24] guarantees on the session data. For the security requirements of this connection, there is likely a minimal impact on overall system security.

For aggregator-mediated communications, mTLS authentication does not present a large burden, since only a single session per aggregator is required. For larger-scale direct-to-device communications, careful consideration on the use of session resumption mechanisms (both for the client and server) may be required to ensure bandwidth and compute resources are used appriopriately. By its nature, session resumption must be optional, and requires support by both client and server – however, when supported, may reduce communications overhead and the costs associated with it.

Note: a number of cloud IoT hubs either have security policies that do not match those required in IEEE 2030.5, or (mutual TLS 1.2 is supported) do not support session resumption. For example, while the Azure IoT hub supports mTLS, it does not support session resumption[25].

7. Conclusion

While the use of dynamic operating envelopes to manage technical risk to network assets shows great promise, it is vital to continue engaging with all stakeholders during design and implementation of these systems. The outcomes of the evolve project can contribute considerably to this process, both in terms of the experience of participants in the implementation of these systems, as well as technical learnings in the design and implementation of the platform.

8. Appendix

API Specification

Please find attached API specification for both versions of the aggregator API deployed in the project.

[24] Perfect forward secrecy describes the assurance that long-term secret compromise does not affect the security of past session keys (and thus the data transferred within past sessions)

[25] See this open support request for further information.

evolve-api-r1 (v0.3.2)

Authentication

registration

Aggregators are provided (out-of-band) a client ID and client secret in order to access the API. These are used within the OAuth 2.0 client credentials flow in order to retrieve an access token with the required scope. The example below (using curl) retrieves the access token with an expiry of 10 hours.

```
curl -X POST \
  http://<host>/o/token/ \
  -H 'Authorization: Basic <base64encoded(client_id:secret)' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d grant_type=client_credentials
```

```
{
    "access_token": "<access_token>",
    "expires_in": 36000,
    "token_type": "Bearer",
    "scope": "read write groups"
}
```

Subsequent requests can be made with the access token as authorization

```
curl -X GET \
  http://<host>/site/site_id/ \
  -H 'Authorization: Bearer <access_token>' \
  -H 'Content-Type: application/json' \
```

Security Scheme Type	OAuth2
clientCredentials OAuth Flow	**Token URL:** http://evolve.cecs.anu.edu.au/o/token/ **Scopes:** • `write` - register devices and write data • `read` - view device and site info

site

End points under the site namespace provide services relating to the registration of sites and devices. A site corresponds to a connection point (NMI), under which distributed energy resources (end devices) may be registered.

While end device information should align to the IEEE 2030.5 standard, the protocol used to communicate this information does not. As such, it is anticipated that these end points will change over time to align more closely with the standard.

Topological information (linking end devices to sites) is not in the scope of 2030.5 but will remain a core concept for this API.

List all available sites

`GET /site/`

AUTHORIZATIONS: registration (`read`)

Responses

— 200 Success

List all the devices for given site

`GET /site/{siteID}/`

AUTHORIZATIONS: registration (`read`)

PATH PARAMETERS

siteID
required
string
Site identifier (NMI)

Responses

— **200** Success

Register a new Site and Devices

AUTHORIZATIONS: registration (`write`)

PATH PARAMETERS

siteID
required
string
Site identifier (NMI)

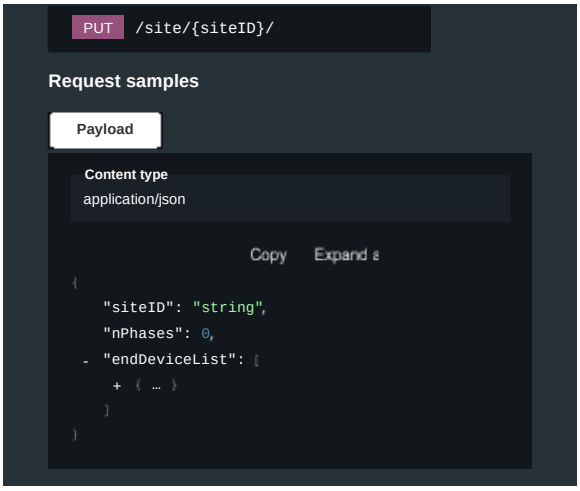
REQUEST BODY SCHEMA: application/json

Site details

siteID
required
string
Site NMI

nPhases
integer
Number of phases

endDeviceList
Array of objects (EndDevice)



Responses

— **200** successful operation

De-register the site from the aggregator

DELETE /site/{siteID}/

AUTHORIZATIONS: registration (`write`)

PATH PARAMETERS

siteID
required
string
NMI to deregister

Responses

— **400** Invalid ID supplied

Register one or more devices at the site using list of devices

AUTHORIZATIONS: registration (`write`)

REQUEST BODY SCHEMA: application/json

List of end devices to be created
Array () [

edevID
required
string
Device identifier

siteID
string
Site ID of linked site. Note that this field is mandatory only when the siteID is not specified either in the URL string or when the device is component of a site in the request body.

deviceCategory
required

integer
Device Category (as per 2030.5-2018 spec). Each bit is '1' if the category is supported by the device

Bit	Category
0	Programmable Communicating Thermostat
1	Strip Heaters
2	Baseboard Heaters
3	Water Heater
4	Pool Pump
5	Sauna
6	Hot Tub
7	Smart Appliance
8	Irrigation Pump
9	Managed Commercial and Industrial Loads
10	Simple Misc (Residential On/Off) Loads
11	Exterior Lighting
12	Interior Lighting
13	Load Control Switch
14	Energy Management System
15	Smart Energy Module
16	Electric Vehicle
17	EVSE
18	Virtual or Mixed DER
19	Reciprocating Engine
20	Fuel Cell
21	PV System
22	Combined Heat and Power
23	Combined PV and Storage
24	Other Generation System
25	Other Storage System

Example “0x800000” - Combined PV and Storage “0x200000” - PV System

deviceReadingAccuracy

integer
Integer describing the accuracy of the device readings

nPhases

integer
Number of phases for device (if applicable)

rtgMaxChargeRateW

number
DERCapability::rtgMaxChargeRateW - the maximum rate (in W) at which the resource can charge.

rtgMaxDischargeRateW

number
DERCapability::rtgMaxDischargeRateW - the maximum rate (in W) at which the resource can discharge.

rtgMaxWh

number
DERCapability::rtgMaxWh - the maximum capacity (in Wh) of the resource.

]

Responses

— **200** Success

Register a new end device at an existing site.

AUTHORIZATIONS: registration (`write`)

PATH PARAMETERS

siteID
required

string
NMI to which device is registered

edevID
required

string
Device identifier

REQUEST BODY SCHEMA: application/json

Device that needs to be added to the site

edevID
required

string
Device identifier

siteID

string
Site ID of linked site. Note that this field is mandatory only when the siteID is not specified either in the URL string or when the device is component of a site in the request body.

deviceCategory
required

integer
Device Category (as per 2030.5-2018 spec). Each bit is '1' if the category is supported by the device

Bit	Category
0	Programmable Communicating Thermostat
1	Strip Heaters
2	Baseboard Heaters
3	Water Heater
4	Pool Pump
5	Sauna
6	Hot Tub
7	Smart Appliance
8	Irrigation Pump
9	Managed Commercial and Industrial Loads
10	Simple Misc (Residential On/Off) Loads
11	Exterior Lighting
12	Interior Lighting
13	Load Control Switch
14	Energy Management System
15	Smart Energy Module
16	Electric Vehicle
17	EVSE
18	Virtual or Mixed DER
19	Reciprocating Engine
20	Fuel Cell
21	PV System
22	Combined Heat and Power
23	Combined PV and Storage
24	Other Generation System
25	Other Storage System

Example “0x800000” - Combined PV and Storage “0x200000” - PV System

deviceReadingAccuracy

integer
Integer describing the accuracy of the device readings

nPhases

integer
Number of phases for device (if applicable)

rtgMaxChargeRateW

number
DERCapability::rtgMaxChargeRateW - the maximum rate (in W) at which the resource can charge.

rtgMaxDischargeRateW

number
DERCapability::rtgMaxDischargeRateW - the maximum rate (in W) at which the resource can discharge.

rtgMaxWh

number
DERCapability::rtgMaxWh - the maximum capacity (in Wh) of the resource.

Responses

— **200** successful operation

Delete the specified device on that site

`DELETE` `/site/{siteID}/edev/{edevID}/`

AUTHORIZATIONS: registration (`write`)

PATH PARAMETERS

- `siteID` (required) — string — NMI to which device is registered
- `edevID` (required) — string — Serial Num of Device

Responses

— **200** successful operation

edev

Device registration and monitoring

Post a list of one or more devices

`POST` `/edev/`

AUTHORIZATIONS: registration (`write`)

REQUEST BODY SCHEMA: application/json

Device to be registered to a site

```
Array () [
```

- `edevID` (required) — string — Device identifier
- `siteID` — string — Site ID of linked site. Note that this field is mandatory only when the siteID is not specified either in the URL string or when the device is component of a site in the request body.

deviceCategory
required

integer

Device Category (as per 2030.5-2018 spec). Each bit is '1' if the category is supported by the device

Bit	Category
0	Programmable Communicating Thermostat
1	Strip Heaters
2	Baseboard Heaters
3	Water Heater
4	Pool Pump
5	Sauna
6	Hot Tub
7	Smart Appliance
8	Irrigation Pump
9	Managed Commercial and Industrial Loads
10	Simple Misc (Residential On/Off) Loads
11	Exterior Lighting
12	Interior Lighting
13	Load Control Switch
14	Energy Management System
15	Smart Energy Module
16	Electric Vehicle
17	EVSE
18	Virtual or Mixed DER
19	Reciprocating Engine
20	Fuel Cell
21	PV System
22	Combined Heat and Power
23	Combined PV and Storage
24	Other Generation System
25	Other Storage System

Example “0x800000” - Combined PV and Storage “0x200000” - PV System

deviceReadingAccuracy

integer

Integer describing the accuracy of the device readings

nPhases

integer

Number of phases for device (if applicable)

rtgMaxChargeRateW

number

DERCapability::rtgMaxChargeRateW - the maximum rate (in W) at which the resource can charge.

rtgMaxDischargeRateW

number

DERCapability::rtgMaxDischargeRateW - the maximum rate (in W) at which the resource can discharge.

rtgMaxWh

number

DERCapability::rtgMaxWh - the maximum capacity (in Wh) of the resource.

Responses

— **201** Device created

— **401** Unauthorized

Get the attributes of the device

`GET` `/edev/{edevID}/`

AUTHORIZATIONS: registration (`read`)

PATH PARAMETERS

- `edevID` (required) — string — Device identifier

Responses

— **200** successful operation

Update the attributes of the device

AUTHORIZATIONS: registration (`write`)

PATH PARAMETERS

- `edevID` (required) — string — Device identifier

REQUEST BODY SCHEMA: application/json

Device that needs to be added to the site

- `edevID` (required) — string — Device identifier
- `siteID` — string — Site ID of linked site. Note that this field is mandatory only when the siteID is not specified either in the URL string or when the device is component of a site in the request body.

deviceCategory
required

integer

Device Category (as per 2030.5-2018 spec). Each bit is '1' if the category is supported by the device

Bit	Category
0	Programmable Communicating Thermostat
1	Strip Heaters
2	Baseboard Heaters
3	Water Heater
4	Pool Pump
5	Sauna
6	Hot Tub
7	Smart Appliance
8	Irrigation Pump
9	Managed Commercial and Industrial Loads
10	Simple Misc (Residential On/Off) Loads
11	Exterior Lighting
12	Interior Lighting
13	Load Control Switch
14	Energy Management System
15	Smart Energy Module
16	Electric Vehicle
17	EVSE
18	Virtual or Mixed DER
19	Reciprocating Engine
20	Fuel Cell
21	PV System
22	Combined Heat and Power
23	Combined PV and Storage
24	Other Generation System
25	Other Storage System

Example “0x800000” - Combined PV and Storage “0x200000” - PV System

deviceReadingAccuracy

integer

Integer describing the accuracy of the device readings

nPhases

integer

Number of phases for device (if applicable)

rtgMaxChargeRateW

number

DERCapability::rtgMaxChargeRateW - the maximum rate (in W) at which the resource can charge.

rtgMaxDischargeRateW

number

DERCapability::rtgMaxDischargeRateW - the maximum rate (in W) at which the resource can discharge.

rtgMaxWh

number

DERCapability::rtgMaxWh - the maximum capacity (in Wh) of the resource.

Responses

— **200** Successfully updated

mup

Sending measurement data

Returns last 'limit' worth of site readings based on 'start' attribute in descending order.'

`GET` `/mup/site/{siteID}/`

PATH PARAMETERS

`siteID` required	string	Site for which measurements were taken

QUERY PARAMETERS

`limit`	string	Number of measurements to return (default: 10, max: 100).

Responses

> **200** Success

Write a set of site and device measurements

`POST` `/mup/site/{siteID}/`

AUTHORIZATIONS: registration (`write`)

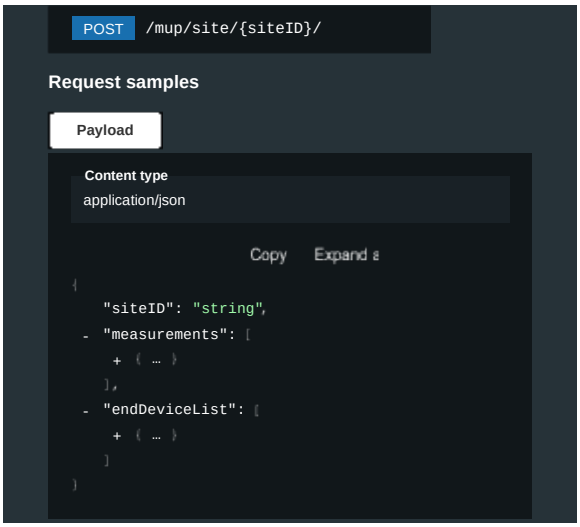
PATH PARAMETERS

`siteID` required	string	Site for which measurements were taken

REQUEST BODY SCHEMA: application/json

Site and device measurements object

`siteID` required	string	
`measurements` ›	Array of objects (Measurement)	
`endDeviceList` ›	Array of objects	



Responses

— **200** Success

Returns last 'limit' worth of end-device readings based on 'start' attribute in descending order.'

`GET` `/mup/site/{siteID}/edev/{edevid}`

PATH PARAMETERS

`edevID` required	string	Device for which measurements were taken

QUERY PARAMETERS

`limit`	string	Number of measurements to return (default: 10, max: 100).

Responses

> **200** Success

envelopes

Endpoints under this namespace serve operating envelopes at a site-by-site level. There are two types of operating envelope schemas based on varied control schemes or modes. These can be used interchangeably for a given timeslot. The two control modes are:

1. RealPower (RP): This mode applies simple real power constraints for import and export, regardless of the value of reactive power.
2. ApparentPower (AP): This mode expands on the RP mode definition with an additional reactive power support ratio, which determines the reactive power required in order to import or export beyond the active power constraint. This allows a system with controllable power factor to compensate for increased import or export by adjusting the system reactive power. An additional parameter, the apparent power constraint, limits the extent to which reactive power support can be used to increase the power transfer at the site.

   NB. All power values are defined in Watts, and use load convention (negative indicating export).

List all operating envelopes for the specified site

AUTHORIZATIONS: registration (`read`)

PATH PARAMETERS

`siteID` required	string Site for which measurements were taken

REQUEST BODY SCHEMA: application/json

Operating envelopes object
`Array () [`

`start` required	integer Start DateTime for the operating envelope in Unix time with second precision.
`duration` required	integer Duration of the operating envelope in seconds.
`phase` required	string Target phase for the operating envelope.
`mode` required	string Enum: `"RP"` `"AP"` The control mode of the given operating envelope.
`published_at` required	integer The Unix timestamp (second precision) when the operating envelope was published.
`importActivePowerConstraint`	number An AP and RP mode property. This is the constraint on the imported real power with no power factor adjustment.
`exportActivePowerConstraint`	number An AP and RP mode property. This is the constraint on the exported real power with no power factor adjustment.
`importReactivePowerSupportRatio`	number An AP mode property. Ratio of the allowable change in imported real power beyond the importActivePowerConstraint that a unit change in reactive power can support.
`exportReactivePowerSupportRatio`	number An AP mode property. Ratio of the allowable change in exported real power beyond the exportActivePowerConstraint that a unit change in reactive power can support.
`importApparentPowerConstraint`	number An AP mode property. This is the constraint on the apparent power value that the given site can import.
`exportApparentPowerConstraint`	number An AP mode property. This is the constraint on the apparent power value that the given site can export.

`]`

Responses

> **200** Success

evolve-api-r2 (0.1.0-beta)

IEEE 2030.5-2018 based API

default_der_control

Read Dderc

`GET` `/edev/{id1}/derp/{id2}/dderc`

Responds with the DefaultDERControl object.

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dderc

`HEAD` `/edev/{id1}/derp/{id2}/dderc`

Responds with the DefaultDERControl object.

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

---

der

Read All Der

Responds with DERList object.

PATH PARAMETERS

- id1 (required) — integer (Id1)

QUERY PARAMETERS

- s — integer (s)
  Default: 0
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.
- l — integer (l)
  Default: 1
  (“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

- accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Der

POST /edev/{id1}/der

PATH PARAMETERS

- id1 (required) — integer (Id1)

HEADER PARAMETERS

- accept — string (Accept)

Responses

> **201** Successful Response

> **422** Validation Error

Read All Der

PATH PARAMETERS

- id1 (required) — integer (Id1)

QUERY PARAMETERS

s — integer (s)
Default: 0
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

l — integer (l)
Default: 1
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Der

`GET /edev/{id1}/der/{id2}`

Response with DER object.

PATH PARAMETERS

id1 (required) — integer (Id1)

id2 (required) — integer (Id2)

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Der

Returns a single DER resource.

Args: id1: Path parameter, the target EndDevice registration number. id2: Path parameter, the DER registration number. accept: HTTP Accept header.

Returns: fastapi.Response object.

PATH PARAMETERS

id1 (required) — integer (Id1)

id2 (required) — integer (Id2)

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Der Capability

`GET` `/edev/{id1}/der/{id2}/dercap`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Der Capability

`PUT` `/edev/{id1}/der/{id2}/dercap`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

Responses

> **201** Successful Response

> **422** Validation Error

Read Der Capability

`HEAD` `/edev/{id1}/der/{id2}/dercap`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dera

`GET` `/edev/{id1}/der/{id2}/dera`

PATH PARAMETERS

- `id1` required — integer (Id1)
- `id2` required — integer (Id2)

HEADER PARAMETERS

- `accept` required — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Dera

`PUT` `/edev/{id1}/der/{id2}/dera`

PATH PARAMETERS

- `id1` required — integer (Id1)
- `id2` required — integer (Id2)

Responses

> **201** Successful Response

> **422** Validation Error

Read Dera

`HEAD` `/edev/{id1}/der/{id2}/dera`

PATH PARAMETERS

- `id1` required — integer (Id1)
- `id2` required — integer (Id2)

HEADER PARAMETERS

- `accept` required — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Ders

`GET` `/edev/{id1}/der/{id2}/ders`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` *required* string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Ders

`PUT` `/edev/{id1}/der/{id2}/ders`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

Responses

> **201** Successful Response

> **422** Validation Error

Read Ders

`HEAD` `/edev/{id1}/der/{id2}/ders`

PATH PARAMETERS

- `id1` *required* integer (Id1)
- `id2` *required* integer (Id2)

HEADER PARAMETERS

- `accept` *required* string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Derg

Returns the DERSettings object.

Args: id1: Path parameter, the end device registration number. id2: Path parameter, the DER registration number. accept: HTTP Accept header.

Returns: fastapi.Response object.

PATH PARAMETERS

id1 required	integer (Id1)
id2 required	integer (Id2)

HEADER PARAMETERS

accept	string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Derg

Creates a DERSettings resource.

Args: id1: Path parameter, the end device registration number. id2: Path parameter, the DER registration number. payload: The request payload/body object.

Returns: fastapi.Response object.

PATH PARAMETERS

id1 required	integer (Id1)
id2 required	integer (Id2)

Responses

> **201** Successful Response

> **422** Validation Error

Read Derg

HEAD /edev/{id1}/der/{id2}/derg

Returns the DERSettings object.

PATH PARAMETERS

- id1 (required) — integer (Id1)
- id2 (required) — integer (Id2)

HEADER PARAMETERS

- accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

---

der_control_response

Create Der Control Response

`POST /rsps/1/rsp`

HEADER PARAMETERS

- content-type — string (Content-Type)

Responses

> **200** Successful Response

> **422** Validation Error

---

der_control

Read Derc List

Responds with the DERControlist object.

PATH PARAMETERS

- id1 (required) — integer (Id1)
- id2 (required) — integer (Id2)

QUERY PARAMETERS

- s — integer (s)
  Default: `0`
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

l — integer (l)
Default: 1
("limit") is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Derc List

Responds with the DERControlist object.

PATH PARAMETERS

id1 — required — integer (Id1)

id2 — required — integer (Id2)

QUERY PARAMETERS

s — integer (s)
Default: 0
("start") is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list's ordering.

l — integer (l)
Default: 1
("limit") is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Derc

`GET /edev/{id1}/derp/{id2}/derc/{id3}`

Responds with the DERControl object.

PATH PARAMETERS

id1 — required — integer (Id1)

id2 — required — integer (Id2)

id3 — required — integer (Id3)

HEADER PARAMETERS

- accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Derc

`HEAD` `/edev/{id1}/derp/{id2}/derc/{id3}`

Responds with the DERControl object.

PATH PARAMETERS

- id1 — required — integer (Id1)
- id2 — required — integer (Id2)
- id3 — required — integer (Id3)

HEADER PARAMETERS

- accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

---

der_curve

Read Dc List

Responds with the DERCurveList object.

PATH PARAMETERS

- id1 — required — integer (Id1)
- id2 — required — integer (Id2)

QUERY PARAMETERS

- s — integer (s)
  Default: `0`
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.
- l — integer (l)
  Default: `1`
  (“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dc List

Responds with the DERCurveList object.

PATH PARAMETERS

id1 *required* — integer (Id1)

id2 *required* — integer (Id2)

QUERY PARAMETERS

s — integer (s)
Default: `0`
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

l — integer (l)
Default: `1`
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dc

`GET /edev/{id1}/derp/{id2}/dc/{id3}`

Responds with the DERCurve object.

PATH PARAMETERS

id1 *required* — integer (Id1)

id2 *required* — integer (Id2)

id3 *required* — integer (Id3)

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dc

`HEAD` `/edev/{id1}/derp/{id2}/dc/{id3}`

Responds with the DERCurve object.

PATH PARAMETERS

- `id1` required — integer (Id1)
- `id2` required — integer (Id2)
- `id3` required — integer (Id3)

HEADER PARAMETERS

- `accept` — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

---

der_program

Read Derp List

Responds with the DERProgramList object.

PATH PARAMETERS

- `id1` required — integer (Id1)

QUERY PARAMETERS

- `s` — integer (s)
  Default: `0`
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.
- `l` — integer (l)
  Default: `1`
  (“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

- `accept` — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

Read Derp List

Responds with the DERProgramList object.

PATH PARAMETERS

- id1 (required) — integer (Id1)

QUERY PARAMETERS

- s — integer (s)
  Default: `0`
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.
- l — integer (l)
  Default: `1`
  (“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

- accept — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

Read Derp

`GET /edev/{id1}/derp/{id2}`

Responds with the DERProgram object.

PATH PARAMETERS

- id1 (required) — integer (Id1)
- id2 (required) — integer (Id2)

HEADER PARAMETERS

- accept — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

Read Derp

`HEAD` `/edev/{id1}/derp/{id2}`

Responds with the DERProgram object.

PATH PARAMETERS

- `id1` integer (Id1) — required
- `id2` integer (Id2) — required

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

dcap

Read Dcap

`GET` `/dcap`

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Dcap

`HEAD` `/dcap`

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

edev

Read Devinfo

`GET` `/edev/{id1}/di`

PATH PARAMETERS

id1 *required* integer (Id1)

HEADER PARAMETERS

accept string (Accept)

Responses

200 Successful Response

422 Validation Error

Set Devinfo

`PUT` `/edev/{id1}/di`

PATH PARAMETERS

id1 *required* integer (Id1)

Responses

200 Successful Response

422 Validation Error

Read Devinfo

`HEAD` `/edev/{id1}/di`

PATH PARAMETERS

id1 *required* integer (Id1)

HEADER PARAMETERS

accept string (Accept)

Responses

200 Successful Response

422 Validation Error

Read Edev List

Responds with a list of EndDevice resources.

QUERY PARAMETERS

s — integer (s)
Default: `0`
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

a — integer (a)
Default: `0`
(“after”) is used to indicate that only items whose primary key occurs after the given date/time (unix epoch) parameter should be included in the query result list.

l — integer (l)
Default: `1`
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Create Edev

`POST /edev`

An EndDevice instance is generated and with its location being defined in the response's 'Location' header.

HEADER PARAMETERS

content-type — string (Content-Type)

Responses

> **201** Successful Response

> **422** Validation Error

Read Edev List

Responds with a list of EndDevice resources.

QUERY PARAMETERS

s — integer (s)
Default: `0`
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

a — integer (a)
Default: `[0]`
(“after”) is used to indicate that only items whose primary key occurs after the given date/time (unix epoch) parameter should be included in the query result list.

l — integer (l)
Default: `1`
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

200 Successful Response

422 Validation Error

Read Edev

`GET /edev/{id1}`

Responds with an EndDevice object.

PATH PARAMETERS

id1 *required* — integer (Id1)

HEADER PARAMETERS

accept — any (Accept)

Responses

200 Successful Response

422 Validation Error

Read Edev

`HEAD /edev/{id1}`

PATH PARAMETERS

id1 *required* — integer (Id1)

HEADER PARAMETERS

accept — any (Accept)

Responses

200 Successful Response

422 Validation Error

Get Rg

`GET /edev/{id1}/rg`

Responds with Registration object.

PATH PARAMETERS

id1
required
integer (Id1)

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful
Response

> **422** Validation
Error

Get Rg

HEAD /edev/{id1}/rg

PATH PARAMETERS

id1
required
integer (Id1)

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful
Response

> **422** Validation
Error

---

mup

Read Mup List

Response with MirrorUsagePointList object.

QUERY PARAMETERS

s
integer (s)
Default: `0`
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

l
integer (l)
Default: `1`
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful
Response

› **422** Validation Error

Create Mup

`POST` `/mup`

Create a MirrorUsagePoint object, path to the instance is defined in the response's 'Location' header.

HEADER PARAMETERS

- `content-type` *required* — string (Content-Type)

Responses

› **200** Successful Response

› **422** Validation Error

Read Mup List

QUERY PARAMETERS

- `s` — integer (s)
  Default: `0`
  (“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.
- `l` — integer (l)
  Default: `1`
  (“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

- `accept` — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

Update Mup

`PUT` `/mup/{id1}`

Update a MirrorUsagePoint.

PATH PARAMETERS

- `id1` *required* — integer (Id1)

HEADER PARAMETERS

- `content-type` *required* — string (Content-Type)

Responses

> **200** Successful Response

> **422** Validation Error

Create Mmr

`POST` `/mup/{id1}`

Create a MirrorMeterReading object.

PATH PARAMETERS

- `id1` integer (Id1) — required

Responses

> **200** Successful Response

> **422** Validation Error

Delete Single Mup

`DELETE` `/mup/{id1}`

Delete a MirrorUsagePoint

PATH PARAMETERS

- `id1` integer (Id1) — required

Responses

— **204** Successful Response

> **422** Validation Error

---

tm

Get Time Resource

`GET` `/tm`

Responds with a Time object

HEADER PARAMETERS

- `accept` string (Accept)

Responses

> **200** Successful

› **422** Validation Error

Get Time Resource

`HEAD` `/tm`

HEADER PARAMETERS

accept — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

fsa

Read Fsa List

Responds with the FunctionSetAssignmentsList object.

PATH PARAMETERS

id1 — integer (Id1)
required

QUERY PARAMETERS

s — integer (s)
Default: `0`
(“start”) is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list’s ordering.

l — integer (l)
Default: `1`
(“limit”) is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept — string (Accept)

Responses

› **200** Successful Response

› **422** Validation Error

Read Fsa List

Responds with the FunctionSetAssignmentsList object.

PATH PARAMETERS

id1
required
integer (Id1)

QUERY PARAMETERS

s
integer (s)
Default: 0
("start") is used to indicate the first ordinal position in the list to be returned in the query result list as determined by the list's ordering.

l
integer (l)
Default: 1
("limit") is used to set the maximum number of list items to be included in the query result list.

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Fsa

`GET` `/edev/{id1}/fsa/{id2}`

Responds with the FunctionSetAssignments object.

PATH PARAMETERS

id1
required
integer (Id1)

id2
required
integer (Id2)

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error

Read Fsa

`HEAD` `/edev/{id1}/fsa/{id2}`

Responds with the FunctionSetAssignments object.

PATH PARAMETERS

id1
required
integer (Id1)

id2
required
integer (Id2)

HEADER PARAMETERS

accept
string (Accept)

Responses

> **200** Successful Response

> **422** Validation Error